NPS-CS-08-001

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

Innovations for Requirements Engineering

by

Luqi & C. Martell

01 January 2008

Approved for public release; distribution is unlimited

Prepared for: National Science Foundation & DARPA
3701 N. Fairfax Drive
Arlington, VA 22203-1714

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver
President

Leonard A. Ferrari
Provost

This report was prepared for National Science Foundation, DARPA, NPS
and funded by National Science Foundation and DARPA.

This report was prepared by:

_____
Luqi
Professor
Computer Science

_____
Craig Martell
Associate Professor
Computer Science

Reviewed by:

_____
Peter Denning
Chair of Computer Science

Released by:

_____
Dan C. Boger
Interim Associate Provost and
Dean of Research

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** January 2008	**3. REPORT TYPE AND DATES COVERED** Technical Report	
4. TITLE AND SUBTITLE: Innovations for Requirements Engineering			**5. FUNDING NUMBERS** **DARPA** **07-W673**
6. AUTHOR(S) Luqi and Craig Martell			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-08-001
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) DARPA 3701 N. FAIRFAX DRIVE ARLINGTON, VA. 22203-1714			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT (maximum 200 words)

The objective of the 15 Monterey workshops since 1992 has been to "increase the practical impact of the formal methods in computer-aided software development". The workshops seek to improve software practice via the application of engineering theory and to encourage development of engineering theory that is well suited for this purpose.

The 2007 workshop focused on requirements, particularly the process of transforming vague and uncoordinated needs of individual stakeholders into consistent and well-defined requirements that are suitable for supporting automated and computer-aided methods for engineering subtasks in the subsequent development process.

Innovations are effective technology transfers of sound inventions. The workshop case study was targeted at identification and assessment of sound inventions of technology that can be used to support innovations in requirement engineering. For example, we wanted to gain a better understanding about how to deal with natural language as the vehicle from which we derive system/software requirements, how to use intelligent agents as entities to facilitate semi-automatic requirements-documentation analysis, and how to build automatic systems to aid in requirements/specifications elicitation. The overall aim was to exchange ideas for continued research in the intersection of these two areas and to reduce the gap between theory and practice.

14. SUBJECT TERMS Requirements Engineering, Software Engineering, National Language Processing, Requirements Document Processing, System Engineering			**15. NUMBER OF PAGES** 204
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

ABSTRACT

The objective of the 15 Monterey workshops since 1992 has been to "increase the practical impact of the formal methods in computer-aided software development". The workshops seek to improve software practice via the application of engineering theory and to encourage development of engineering theory that is well suited for this purpose.

The 2007 workshop focused on requirements, particularly the process of transforming vague and uncoordinated needs of individual stakeholders into consistent and well-defined requirements that are suitable for supporting automated and computer-aided methods for engineering subtasks in the subsequent development process.

Innovations are effective technology transfers of sound inventions. The workshop case study was targeted at identification and assessment of sound inventions of technology that can be used to support innovations in requirement engineering. For example, we wanted to gain a better understanding about how to deal with natural language as the vehicle from which we derive system/software requirements, how to use intelligent agents as entities to facilitate semi-automatic requirements-documentation analysis, and how to build automatic systems to aid in requirements/specifications elicitation. The overall aim was to exchange ideas for continued research in the intersection of these two areas and to reduce the gap between theory and practice.

Monterey Workshop 2007

PRE-PROCEEDINGS of the 14th Monterey Workshop on Innovations for Requirements Analysis: From Stakeholders Needs to Formal Design

September 10-13, 2007, Monterey, CA, USA

Workshop Chairs:
Luqi Naval Postgraduate School, Monterey, California, USA
Fabrice Kordon University of Pierre & Marie Curie, Paris, France

Workshop Website: http://fabrice.kordon.free.fr/Monterey2007/
Workshop mail: montereyworkshop@gmail.com

Sponsored by DARPA, NSF, NPS

Steering Committee
David Hislop - Army Research Office, USA
Luqi - Naval Postgraduate School, Monterey, California, USA
Zohar Manna - Stanford University, California, USA
Manfred Broy - Technical University Munich, Germany
Egidio Astesiano - University of Genova, Italy
Fabrice Kordon - University of Pierre & Marie Curie, Paris, France
Janos Sztipanovits - Vanderbilt University, Nashville, Tennessee, USA
Hermann Kopetz - Vienna University of Technology, Vienna, Austria

Program Co-Chairs
Craig Martell - Naval Postgraduate School Monterey, USA
Barbara Paech - University of Heidelberg, Germany

Program Committee
Daniel M. Berry - University Waterloo, Canada
Christine Choppy - University Paris XIII, France
Stephen Clark - Oxford University, UK
Lori A. Clarke - University of Massachusetts, USA
Rance Cleveland - University of Maryland, USA
Vincenzo Gervasi - University of Pisa, Italy
Aravind Joshi - University of Pennsylvania, USA
Kane Kim - University of California, Irvine, USA
Leonid Kof -Technical University of Munich, Germany
Fabrice Kordon - University P. & M. Curie, France
Bernd J. Krämer, FernUniversität Hagen, Germany
Mitch Marcus - University of Pennsylvania, USA
Bashar A. Nuseibeh - The Open University, UK
Manuel Rodriguez - National Research Council, USA
Sol Shatz - University of Illinois at Chicago, USA
Chen-Yu Phillip Sheu - University of California, Irvine, USA

Local Committee
Craig Martell - Naval Postgraduate School Monterey, USA

Compilation of the Pre-Proceedings
Willi Springer - University of Heidelberg, Germany

Support Staff
Daniel Cook - Naval Postgraduate School Monterey, USA
Melissa Fan - Naval Postgraduate School Monterey, USA
Kathleen Farrell - Naval Postgraduate School Monterey, USA
Christine Kays - Naval Postgraduate School Monterey, USA
Valerie Ramirez - Naval Postgraduate School Monterey, USA
James Stapleton - Naval Postgraduate School Monterey, USA
Keisha Williamson - Naval Postgraduate School Monterey, USA

Table of Contents:

Thursday 13 September

Position Papers

Innovations for Requirements Analysis: From Stakeholders Needs to Formal Designs

Luqi & Fabrice Kordon
Monterey Workshop 2007 Chairs

1. Introduction to the Monterey Workshop

The objective of the entire series of 15 Monterey workshops since 1992 has been to "increase the practical impact of formal methods in computer-aided software development". The workshop seeks to improve software practice via application of engineering theory and to encourage development of engineering theory that is well suited for this purpose.

Previous workshops have reduced the gap between theoretical and practical aspects of software/system engineering and have produced a consensus that the pain of system development could be reduced via computer aid for or automation of software engineering subtasks based on particular theories and various kinds of formal models. A common theme has been to hide theoretical results and complex mathematical ideas inside tools with simple interfaces so that practitioners could use them without the need to fully understand the theory behind them.

However, there has also been general agreement that the pain of development cannot be eliminated completely. No matter what you do, somewhere in the process some people have to think clearly and in detail to reach agreement on what problems should be solved by the software to be developed. Consequently, requirements, response to changes, and human aspects of programming have been identified as potentially fruitful areas for improvement.

2. Goal of Monterey Workshop 2007

The 2007 workshop is focused on requirements, particularly the process of transforming vague and uncoordinated needs of individual stakeholders into consistent and well defined requirements that are suitable for supporting automated and computer aided methods for engineering subtasks in the development process to follow.

Errors or failures of software-based systems are due to a variety of causes, e.g. misunderstanding of the real world, erroneous conceptualization, or problems in representing concepts via the specification or modeling notations. Precise specification is a key success factor as are communication and the deliberation about whether the specification is right and whether it has been properly implemented. Not all stakeholders are familiar with the formal models and notations employed. Some important requirements might be difficult to quantify and/or express using formal languages, such as the desire that a system should be user-friendly or easily maintainable. Better technologies for requirements analysis should be thus considered.

The majority of requirements are given in natural language, either written or orally expressed. Other requirements might also be visually expressed in terms of figures, diagrams, images or even gestures. Artificial-intelligence approaches might be used to develop prototypes, which can then be re-engineered using more conventional requirements technologies and safety assurance techniques. For example, we might employ large amounts of semantic and statistical data, knowledge bases and theorem provers to infer as much contextual information as possible from the (vague) textual or visual requirements. Then, some extra questions could be raised to system/software stakeholders to point out some fuzzy (or missing) requirements to be refined or some conflicting requirements to be reconciled.

The automatic analysis of natural language expressions has not yet been fully achieved, and interdisciplinary methodologies and tools are needed to successfully go from natural language to accurate formal specifications. Conformance of a system implementation to its requirements requires dynamic and efficient communication and iteration among system stakeholders. It is in supporting this process, and not in supplanting it, that innovative approaches to requirements analysis need to find their proper role.

We want to gain a better understanding about how to deal with natural language as the vehicle from which we derive system/software requirements, how to use intelligent agents as entities to facilitate semi-automatic requirements-documentation analysis, and how to build automatic systems to aid in requirements/specifications elicitation. The overall aim is to exchange ideas for continued research in the intersection of these two areas and to reduce the gap between theory and practice.

A good case study for these issues is to consider how to extract a conceptual model of the goals and requirements of the software needs discussed in a *blog*. As blogs are unstructured natural language, they represent one of the most difficult challenges for natural language processing. All workshop participants have been requested to use the case study given in Section 4 of this paper to illustrate their work.

3. Focus Areas

The three days of the workshop are organized around the following focus areas:

Recent Advances in Requirements Engineering. Feather compares various approached to specification and requirements analysis. Aschauer et al explore success factors for agile requirements analysis. Dinesh et al present an approach for regulatory conformance checking.

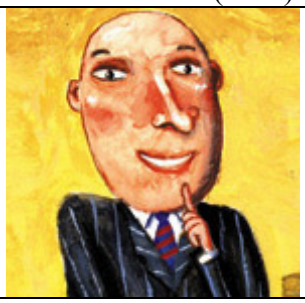
Human and Linguistic aspects of Requirements Engineering. Kof addresses identification of goals in stakeholder dialogs. Sawyer examines profiling and tracing of stakeholder needs. Goedecke explores the relation between viewpoints and documents.

Computer Aid for Requirements Engineering. Fe describes how model-driven prototyping can help elicit requirements. Popescu et al explain how automatically created OO models can be used to improve the quality of requirements specifications.

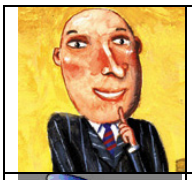
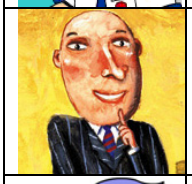
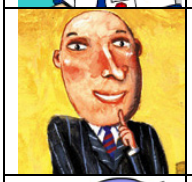
The panels and discussion sections interleaved with the presentations are focused on integrating, balancing, and assessing the various viewpoints presented at the workshop to reach a consensus on where we are, how emerging capabilities for natural language processing and computer aided requirements elicitation methods can contribute, and to identify the best paths forward.

4. Workshop Case Study

All workshop participants have been asked to use the case study given hereafter to illustrate their work.

Transportation Security Administration (TSA)	Federal Aviation Administration (FAA)	Airport screening and security
		

Blogger	Post
	We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot. We are also working on technologies to screen for chemicals in liquids, backscatter, you know…
	Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Chanel No. 5 you get False Alarms … so it is not quite ready for deployment!
	Come on! Generating false positives helped us stay alive; maybe that wasn't a lion that your ancestor saw, but it was better to be safe than sorry. Anyway, I want you to be more alert - airport screeners routinely miss guns and knives packed in carry-on luggage.
	Well… It's not easy to move 2 million passengers through U.S. airports daily. And people can't remain alert to rare events, so they slip by.

	We can deal with it. What if you guys take frequent breaks? And also we are going to artificially impose the image of a weapon onto a normal bag in the screening system as a test. Then screeners learn it can happen and must expect it. Eventual solution will be a combination of machine and human intelligence.
	Sounds good though we do take breaks and are getting inspected. We do not get annual 'surprise' tests - sometimes we get them everyday; and if a screener misses too may of these consistently, they are sent to training.
	We have yet to take a significant pro-active step in preventing another attack - everything to this point has been reactive. Somebody hijacks a plane with box cutters? - Ban box cutters. Somebody hides explosives in their shoes? - X-ray shoes, and then ban matches. We are well behind!
	What do you suggest? Yes, there is an uncertainty. On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect. There are no easy solutions. We are trying to federalize checkpoints and to bring in more manpower and technology.
	We need to think ahead. For instance, nobody needs a metal object to bring down an airliner, not even explosives. Practically everything inside the aircraft is easily flammable, except for the people, so all anyone needs is oxidizer. Do any of the automated screening devices detect oxidizers? Are the human screeners trained to recognize them?
	Good point. Airlines need to take the lead on aviation security. The corporate response was to market cheap tickets and pass security off on the federal government. Have a trained group of security officers on every flight. Retrain flight attendants as security officers. Forget about passing around the soda and peanuts - that should be secondary.
	Sir, a lot of airlines are not doing well and are on the Government assistance. Prices go up, baggage get mishandled. There are constant changes in screening rules – liquids/no liquids/3-1-1 rule. Anything radical will not only cost a lot of money but also deter people. I mean an economic threat is also a threat.
	I think that enforcing consistency in our regulations and especially in their application will be a good thing to do. Another thing is that even if an airline goes bankrupt there are still advantages: bankruptcy makes it easier to rearrange company assets and to renegotiate vendor and supplier contracts.
	Ok, we had very productive discussion. Now back to work. I want you to come up with some concrete measures based on what we have been talking about. You should finally generate some ROI for that money we have been spending. And do not forget, the examples listed above are not all-inclusive.

The objective of the case study exercise is to answer the following questions based on the discussion above:

1. What was the topic(s) of the discussion? Have you noticed any contradictions?
2. What are the realistic requirements that FAA suggests for increasing airport security?
3. What long-terms goals should be set by TSA?
4. What concrete changes should be enforced by Airport screening and security?

5. Conclusion

Overarching goals of the rest of the series of Monterey Workshops are to create a shared community-wide articulation of the system/software engineering enablement challenge, reach consensus on the set of intellectual problems to be solved, and create a common vision of how the solutions to these problems will fit together in a comprehensive engineering environment.

The Monterey Workshop has been able to bring the brightest minds in Software Engineering together with the purpose of increasing the practical impact of formal methods for software development so that these potential benefits can be realized in actual practice. In the workshop, attendees and organizers work to clarify what good formal methods are, what are their feasible capabilities, and what are their limits. Overall, the workshop strives to reduce the gap between theory and practice. This has been a slow and difficult process because theoreticians and practitioners do not normally talk to each other, and did not at the beginning of the workshops. This gap has been gradually reduced. In particular, researchers have focused on problems that are relevant to the practitioners, and have helped demonstrate how recent theory can be applied to solve current problems in software development practice.

Here are the workshops:

N	Year	Theme	Location	Chairs
0	1992	Concurrent and Real-Time Systems	Monterey	Luqi, Gunter
1	1993	Software Slicing, Merging and Integration	Monterey	Berzins
2	1994	Software Evolution	Monterey	Luqi, Brockett
3	1995	Specification-Based Software Architecture	Monterey	Luqi
4	1996	Computer-Aided Prototyping	Monterey	Luqi
5	1997	Requirements Targeting Software and System Engineering	Bernried	Broy, Luqi
6	1998	Engineering Automation for Computer-Based Systems	Carmel	Luqi, Broy
7	2000	Modeling Software System Structures in a Fastly Moving Scenario	Santa Margherita Ligure	Astesiano, Broy, Luqi
8	2001	Engineering Automation for Software Intensive System Integration	Monterey	Luqi, Broy

9	2002	Radical Innovations of Software and Systems Engineering in the Future	Venice	Wirsing
10	2003	Embedded Systems	Chicago	Shatz
11	2004	Compatibility and Integration of Software Engineering Tools	Vienna	Manna, Henzinger
12	2005	Networked Systems	Irvine	Sztipanovits, Kordon
13	2006	Composition of Embedded Systems	Paris	Kordon, Sokolsky
14	2007	Innovations for Requirements Analysis	Monterey	Luqi, Kordon
15	2008	Foundations of Computer Software	Budapest	Sztipanovits

These workshops have helped focus the attention of the community on many productive directions. For example, since the 1995 workshop identified specification-based architectures as a key means to achieve system flexibility and reuse, there has been a great deal of activity in these areas. A great deal of research has produced architecture description languages and associated analysis methods, there have been commercial advances on "plug and play" hardware and software, adoption of service-based architectures in electronic commerce, and a move toward open architectures in government and defense systems. Currently the practical impact of software architecture is no longer in doubt.

We look forward to comparable developments on computer aided requirements analysis in the decade to come.

Acknowledgements

The Monterey Workshops were initiated under the support of Dr. Hislop at ARO and many others at NSF, ONR, AFOSR, and DARPA. We would like to thank DARPA and NSF for their financial support of the 2007 workshop, NRC support of two talented postdoctoral fellows Dr. Rodriguez and Dr. Ivanchenko who contributed to the proposal, workshop case study and material for the web page, the program committee chairs Barbara Paech and Craig Martell and committee members for their efforts on reviewing papers and putting together the workshop program, and the local chair Craig Martell for handling endless practical details.

Ambiguity in Natural Language Requirements Documents

Daniel M. Berry
School of Computer Science
University of Waterloo
200 University Ave. West
Waterloo, Ontario N2L 3G1 Canada
dberry@uwaterloo.ca
http://se.uwaterloo.ca/~dberry

Abstract

When requirements are written, as they usually are, in natural language, ambiguity is a major cause of their not specifying what they should and implementers implementing the wrong system. Simple misuse of the language in which the document is written is one source of these ambiguities.

This talk argues that even when formal methods are used, natural language is key in requirements engineering. The talk describes the ambiguity phenomenon from several points of view, including linguistics and software engineering. Several strategies for avoiding and detecting ambiguities are presented. Strong emphasis is given on the problems arising from the use of heavily used and seemingly unambiguous words, phrases, and constructs such as ``all", ``each", `every", and plural in defining or referencing sets; the positioning of ``only" and ``also"; and referents of pronouns. Many examples from requirements documents are examined. The talk concludes with a discussion of requirements for automated tools for finding ambiguities.

Discussion of Some Past and Current Approaches to Specification and to Requirements Analyses

Martin S. Feather[1]

[1] Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Dr,
Pasadena CA 91109-8099
Martin.S.Feather@jpl.nasa.gov

Abstract. Several decades ago Balzer at al aimed to aid users compose precise and correct specifications from informal natural language descriptions. How they approached this problem may be of interest to this workshop. The workshop, however, has a focus on requirements (as distinct from specifications). This suggests a significant difference in the issues that need to be dealt with, even though the end points – natural language, and implementations, might be the same. Some ongoing themes of research are mentioned, leading to a focus on what is clamed as an understudied area, support for requirements decision making on the basis of partial information. Goal graph work is suggested as exemplifying work in this area, and lastly the author's own work in a quantitatively-based goal graph like approach is briefly illustrated with respect to the case study blog.

Keywords:.Specification freedoms, requirements, goals, objectives, quantitative reasoning, risk-informed decision making

1 Early Work on Natural Language to *Specification* to Implementation

In the 1970s Bob Balzer was Project Leader of the Specification Acquisition From Experts (SAFE) Project at the Information Sciences Institute, University of Southern California. Some of this work is described in [1]; quoting from Balzer's biography at the end, "This project is attempting to aid users to compose precise and correct program specifications from informal natural language descriptions by resolving the ambiguity present through context provided by the system's knowledge of programs, programming, and the application domain". One of the key decisions Balzer et al made was to *design a formal specification language that was as close as possible to the way people tended to express problem specifications in natural language*. They began by establishing the principles that such a language should exhibit, documented in [2]. They then developed the language Gist to fulfill these principles [3]. Gist thus served as a stepping stone – a problem could, it was hoped, readily be converted from an informal, natural language, formulation into its formal equivalent in Gist; thereafter, program synthesis and program transformation techniques [4] could be applied to convert that specification into an efficient implementation. These ambitions

led to much work by Balzer's group, described in a retrospective paper [5]. Much of the effort was expended on the step from formal specification to implementation, an area of investigation by several research groups at that time. Challenges remain in that area, as outlined in a more recent retrospective [6]. The step from natural language to formal specification received relatively little attention once the specification language Gist had been developed. Balzer's group did some work in the reverse direction, generating natural language as a means to explain the formal specification, and to explain the results of symbolic evaluation of the formal specification.
With this history in mind, how does this relate to the workshop's focus on *requirements* analysis?

2 Requirements Analysis … Needs to Designs

The workshop's title and statement of objectives include mention of, among other things:

- Requirements analysis: From Stakeholders Needs to Formal Designs
- Precise specification (for deliberation both for "whether the specification is right" and "whether it has been properly implemented")
- Challenging requirements to express formally ("user-friendly", "easily maintainable")
- Inferring contextual information, and aiding in requirements/specifications elicitation

These, together with the blog case study, indicate that while the ultimate goal might be to arrive at a formal specification (or even a design) which will thereafter be implemented, *there is much to do to arrive at what that formal specification should be*. It is this that I interpret as the realm of "requirements analysis". A while back Steve Fickas and I argued that, just as specifications exhibit freedom from implementation concerns (a theme of the work I was involved in as a member of Balzer's group [6]), requirements exhibit freedom from specification concerns [7]. Briefly, we argued that requirements may well exhibit incompleteness, inconsistency, redundancy, ambiguity, non-uniformity, and heterogeneity, and that it is these "freedoms" that have to be removed to arrive at a specification (which will ideally be complete, consistent, non-redundant, unambiguous, uniform, and homogeneous). NOTE: there may be some different interpretation to the term "requirements" – some would argue that they are specification clauses. In the spirit of the work I did with Fickas, and (I think) with the intent of this workshop, I am treating requirements and requirements analysis as encompassing what some might term "goals", "objectives", "desiderata", etc., and fully expect that there will be the need for negotiation, compromise, iteration, decision-making, etc., to deal with them.

With this in as the focus, what does the history of Balzer's specification-centered work suggest? One possibility is to take the same tack: look at how people naturally state requirements in natural language, strive to construct a *requirements* notation that provides a no less convenient yet rigorous equivalent, and develop the tools and techniques that will operate on that language to help perform the analyses,

translations, feedback, etc. Yet scrutiny of the outcome of the history of Balzer et al's efforts suggests that while they inspired a good deal of interesting work, they did not, in the end, traverse the (in retrospect, highly ambitious) pathway all the way from natural language, through specification, to implementation. So a plausible alternative way to proceed would be to focus on the intermediate steps of going from natural language through requirements to specification: examine these steps to clarify their purpose, understand the current state of the art, and extend existing capabilities, or adapt them, or unify them etc, as the case may be, to lead to *incremental* progress.

On the formal or semi-formal representation and analysis aspects, there has been, and continues to be, many examples of work that fits this pattern (I am much less familiar with related work that focuses on the natural language aspects). For example, in the 1990's the "viewpoints" concept was being actively pursued by a number of researchers as a means to represent and deal with multiple stakeholders' differing perspectives on the software system they were trying to ultimately produce (for an overview, see [8]; for a workshop covering this area, [9]). Various studies have focused on activities in what Robinson et al termed "Requirements Interaction Management" [10] ("… the set of activities directed toward the discovery, management, and disposition of critical relationships among sets of requirements"). Much related work continues under the rubric "Goal-Oriented Requirements Engineering", as surveyed in [11] and in a follow-on [12].

So I think the workshop could yield:

- An understanding of how natural language could/should influence the ongoing body of requirements engineering research and application,
- A roadmap to improving the ongoing areas of requirements engineering research – identifying how they should be combined, validated, etc.
- A "gap analysis" to reveal which areas are relatively understudied yet worthwhile, and so in most need of attention. I believe that the work I have been involved in recently at JPL fits into one of these areas.

3 An Understudied Area – Support for Requirements Decision Making on the Basis of Partial Information

At requirements time there is often the need to make influential decisions on the basis of partial information. I assert that support for this is a relatively understudied area. Approaches that assume anything remotely close to complete capture of all relevant information to decision making will nearly always be impractical. Instead, the goal should be to make generally good decisions (albeit at the risk of missing superior solutions, or of needing to perform backtracking to revise previous decisions that turn out to lead to undesirable results). As a consequence, the need to have a "perfect" representation – one that can capture all the nuances and interactions of a problem domain – is much less of a driver than it would be for, say, a representation of a formal specification that will lead to implementation.

The goal graph representations and the uses to which they are put, as seen in the NFR[13] / i*[14] / Tropos [15] lines of work, exemplify a judicious choice of

representation and analysis capabilities. I believe the workshop's case study blog would be interesting to see represented in this style.

During recent years I have been working on an approach (called DDP [16]) that bears some similarity to the above-referenced goal graphs. It has been successfully utilized predominantly to help make technology development decisions relatively early in their lifecycle [17]. In contrast to the goal graphs work, it adopts a more restrictive graph structure, but extensively utilizes numerical calculations to help perform analyses over the accumulated data. My JPL colleague Steve Cornford who invented the DDP approach envisioned from the start the utility of a numerical basis for DDP's calculations. Despite the facts that there are known pitfalls with trying to ascribe and utilize numerical values, our positive experiences suggest that it has indeed helped teams of stakeholders arrive at superior solutions to requirements problems. The subsections that follow introduce the core concepts of DDP, and show (the start of) application of DDP to the workshop case study.

Brief introduction to DDP

A DDP model is populated by instances of three kinds of concepts: *Objectives* – what it is that the system or technology under scrutiny is to achieve, *Risks* – what could occur to impede the attainment of the Objectives, and *Mitigations* – what could be done to reduce the likelihood and/or impact of Risks[1]. In the DDP model these instances have quantitative attributes: each Objective has a *weight*, its relative importance; each Risk has a likelihood, its probability of occurrence, and each Mitigation has a cost, the cost of performing it – usually a financial cost, but other resources can also be considered, such as schedule, power, mass. Quantitative relationships connect these instances: Objectives are related to Risks, and Risks to Mitigations. Specifically, Objectives are related to Risks to indicate how much each Risk, should it occur, *impacts* (i.e., detracts from the attainment of) each Objective. Risks are quantitatively related to Mitigations, to indicate how much of a Risk-reducing *effect* a Mitigation, should it be applied, has on reducing each Risk, either by decreasing the Risk's likelihood, or by reducing the magnitude of the Risk's impacts on Objectives; the nature of the Mitigation dictates which kind of reduction takes place.

The primary purpose of constructing a DDP model is to help decide which Mitigations to perform; Mitigations incur costs and, by their Risk-reducing effects, achieve benefits (measured in terms of attainment of Objectives). DDP helps users balance these concerns when choosing Mitigations. A DDP model can also help users decide which Objectives to abandon, should it prove too costly to attain all of them.

The usual way of developing and using a DDP model is to assemble a group of relevant stakeholders in a series of facilitated sessions. The information that goes into the DDP model is elicited from those stakeholders, and thereafter they make decisions on its basis. Custom software has been developed to support application of DDP in

[1] On occasion our publications use alternate terminology such as "Requirements" in place of "Objectives", "Failure Modes" in place of "Risks", and "PACTs" – an acronym for Preventative measures, Analysis, process Controls, and Tests – in place of "Mitigations".

these sessions. A user familiar with DDP "drives" the software, and the DDP screen is projected so as to be visible to all stakeholders. As they proffer information, it is entered into DDP on-the-fly, and the status of the information is visible to all. DDP performs the calculations of cost and benefit as various selections of Mitigations are studied, and offers a form of heuristic search (using simulated annealing) to help locate near-optimal selections of Mitigations (e.g., maximize attainment of Objectives while remaining within some cost limit). DDP employs a variety of visualizations to present the accumulated information [18].

Initial application of DDP to the case study

A hasty first-cut attempt applying DDP to represent some of the information found in the blog is shown next.

DDP's *Objectives* are used to represent desiderata expressed by the "Blogger Participants". At a minimum, DDP requires for each a short title string to serve as a pithy description, e.g.,

"Invulnerable to terrorist plot"
"High passenger throughput"
"Few false positives"
"Avoid baggage mishandling"

Further information may be associated a DDP Objective, such as a lengthier textual description, and the source (i.e., indication of which one of the Bloggers provided the Objective). By default, Objectives are initialized with the same relative weight (1). These weights can be adjusted to represent priorities, e.g., if it is twice as important to achieve the Objective "Invulnerable to terrorist plot" as "High passenger throughput", then the former would be ascribed double the weight of the latter.

DDP's *Risks* are used to represent conditions and events that potentially detract from attainment of Objectives, e.g.,

"Known attack modes"
"New attack modes"
"Screeners inattentiveness"
"False alarms"
"Passengers confused about regulations"

Lengthier textual descriptions, and their sources, may be associated with Risks. By default, each Risk is initialized with an "a-priori likelihood" of 1, meaning the Risk is sure to occur unless Mitigations are applied to inhibit it. Also by default, DDP's Risks are each "atomic" objects, i.e., lack any further structure. It is possible to compose DDP's Risks into fault trees using "And", ""Or" and "Not" gates. For example, false alarms may occur in current practice, and, as suggested in the second posting, can also be caused by new technologies that are "not quite ready for deployment". Thus the "False alarms" risk could be structured into an "Or" of false alarms from conventional screening, and false alarms from new screening technologies.

DDP's *Mitigations* are used to represent the practices that reduce risks, both those in current use, and those suggested as potential improvements, e.g.,

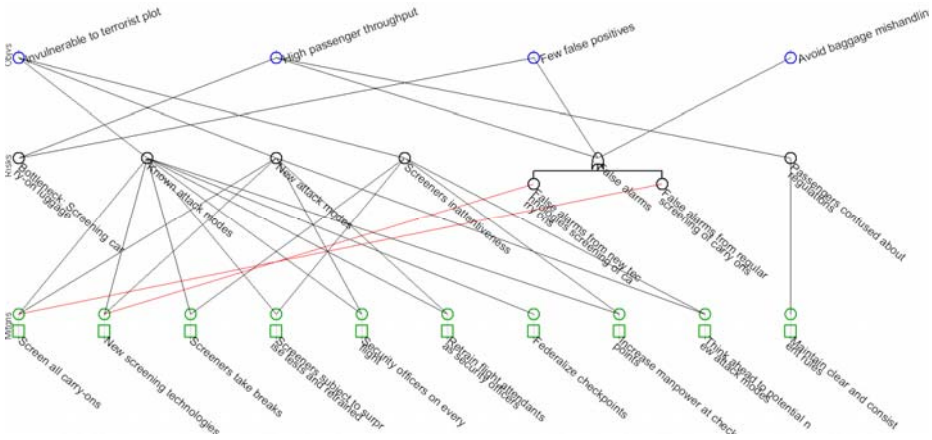
"Ban taking liquids on board"
"Screen all carry-on luggage"

"New screening technologies"
"Screeners take breaks"
"Screeners subject to surprise tests and retraining"
"Security officers on every flight"
"Retrain flight attendants as security officers"
"Federalize checkpoints"

These too can have associated textual descriptions, and sources. As the information becomes available, costs can be associated with Mitigations.

DDP then requires that these concepts be linked – Risks are linked to the Objectives whose attainment they threaten, and Mitigations are linked to the Risks they quell. Furthermore, in order to employ DDP's quantitative reasoning, these links have to be given values; a Risk-Objective link is given an impact value, the proportion of the Objective's attainment that would be lost were the Risk to occur; a Mitigation-Risk link is given an effect value, the proportion by which the Risk is reduced if the Mitigation is applied.

For purposes of illustration, I have entered into DDP some of the Objectives, Risks and Mitigations that I saw within the case study, and made some (as yet unquantified) connections among them. The figure below shows this connectivity (this is a screenshot taken from one of DDP's visualizations).



The top row, of small blue circles, represent Objectives (labeled with their title strings). The bottom row of small green circles and squares represent Mitigations (the squares are actually checkboxes on the DDP screen for displaying and controlling which of the Mitigations are chosen). The middle row of black circles and a small fault tree structure with an "Or" gate represent Risks. The lines connecting the Risks to the Objectives indicate which Objectives are impacted by which Risks (absence of a line denotes zero impact), but as yet no numerical values have been ascribed to these connections. The lines connecting the Mitigations to the Risks indicate which Risks are reduced by which Mitigations (absence of a line denotes no effect). Red colored lines indicate Mitigation-Risk connections where the Mitigation makes the Risk *worse*.

It is noteworthy that the case study blog does not contain indications of connectivity among these concepts, and provides no indication as to the magnitude of the various effects. E.g., how much improvement (reduction of risk, and thereby increase in attainment of objectives) will be gained if checkpoints are federalized? What is the cost of doing so? Without such information I assert that it is very hard to make any meaningful decisions. In the style of the goal graph work, *qualitative* valuations would be utilized – I would be interested to see the extent to which this would enable decision making. In the DDP style, *quantitative* valuations are expected, and enable various forms of decision making (e.g., identification of the biggest risks, of the objectives most under threat, of the cost-benefit tradeoffs of different selections of actions). On this basis I assert that the natural language descriptions found in the case study blog are a good start at establishing the concepts and how they interrelate, but the final comment in the blog "come up with some concrete measures" matches the need, in DDP, to ascribe quantitative valuations in order to support decision making.

Acknowledgments. Some of the work described in this paper was performed at Information Sciences Institute, University of Southern California, sponsored by DARPA, and some at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. Balzer, R., Goldman, N., Wile, D.: Informality in Program Specifications. IEEE Transactions on Software Engineering, Vol. SE-4 No 2 (March 1978) 94–103
2. Balzer, R., Goldman, N.: Principles of Good Software Specification and their Implications for Specification Languages. Proc. Specifications of Reliable Software Conference, Boston, MA, April 1979 IEEE Computer Society (1979) 58–67, reprinted in Software Specification Techniques, Gehani, N., McGettrick, A.D., eds., Addison Wesley (1986) 25–39
3. Balzer, R.: Transformational Implementation: An Example. IEEE Transactions on Software Engineering, Vol. SE-7 No 1 (Jan 1981) 3–14
4. Balzer, R.: A 15 Year Perspective on Automatic Programming. IEEE Transactions on Software Engineering, Vol. SE-11 No 11 (November 1985) 1257–1268
5. Wile, D.S.: Panel Topic and Whitepaper: The Lessons of the 80's, Proceedings of a Workshop on Transformation Technology, ICSE '99 (May 1999)
6. London, P.E., Feather, M.S.: Implementing Specification Freedoms. Science of Computer Programming 2 North-Holland (1982) 91–131, reprinted in Readings in Artificial Intelligence and Software Engineering, Morgan Kaufmann (1986) 285–305
7. Feather, M.S., Fickas, S: Coping with Requirement Freedoms. Position paper for the Workshop on Intelligent Information Systems, Niagara-on-the-lake, Ontario, Canada, April 1991 University of Toronto Press, Toronto (1991) 42–46
8. Finkelstein, A., Sommerville, I.: The Viewpoints FAQ. Software Engineering Journal Vol 11 No 1 (1996) 2–4
9. Vidal L., Finkelstein, A., Spanoudakis, G., Wolf, A.L.: Proceedings of the International Workshop on Multiple Perspectives in Software Development, Part II of the Joint Proceedings of the SIGSOFT '96 Workshops, ACM (1996) 155–297

10. Robinson, W.M., Pawlowski, S.D., Volkov, V.: Requirements Interaction Management, ACM Computing Surveys, Vol 35 No 2 (June 2003) 132–190
11. van Lamsweerde, A.: Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice, 2004 IEEE International Requirements Engineering Conference, (2004), 4–7
12. Mylopoulos, J.: Goal-Oriented Requirements Engineering: Part II, Keynote, 14th IEEE Requirements Engineering Conference (2006), http://www.ifi.unizh.ch/req/events/RE06/ConferenceProgram/RE06_slides_Mylopoulos.pdf
13. Mylopoulos, J., Chung, L., Nixon, B.: Representing and Using Non-Functional Requirements : A Process-Oriented Approach, IEEE Transactions on Software Engineering Vol 18 No 6, (June 1992) 483–497
14. Yu, E.: Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, 3rd IEEE International Symposium on Requirements Engineering (Jan 1997), 226–235
15. Giogini, P., Mylopoulos, J., Sebastiani, R.: Goal-Oriented Requirements Analysis and Reasoning in the Tropos Methodology, Engineering Applications of Artificial Intelligence, Vol 18 No 2, (March 2005) 159–171
16. Feather, M.S. and Cornford, S.L. "Quantitative Risk-Based Requirements Reasoning" Requirements Engineering (2003) 8: pp. 248-263.
17. Feather, M.S., Cornford, S.L., Hicks, K.A., Johnson, K.R.: Applications of tool support for risk-informed requirements reasoning, Computer Systems Science and Engineering, Vol 20 No 1 (January 2005) 5–17
18. Feather, M.S., Cornford, Menzies, T., Kiper, J.D.: Experiences using Visualization Techniques to Present Requirements, Risks to Them, and Options for Risk Mitigation, International Workshop on Requirements Engineering Visualization (REV 2006) Minneapolis / St. Paul, MN, Sep 2006

Key factors of a successful agile requirements analysis in the realm of overhauling an industrial automation system

Thomas Aschauer[1], Gerd Dauenhauer[1], Patricia Derler[1], Wolfgang Pree[1],
Christoph Steindl[2]

[1] C. Doppler Laboratory Embedded Software Systems, Univ. Salzburg
Jakob-Haringer-Str. 2, 5020 Salzburg, Austria
firstname.lastname@cs.uni-salzburg.at
www.cs.uni-salzburg.at

[2] Catalysts GmbH,
Prager Str. 6, 4040 Linz, Austria
steindl@catalysts.cc
www.catalysts.cc

Abstract. This paper sketches a recent successful requirements analysis of a complex industrial automation system that mainly required a talented expert, with a beginner's mind, who has been willing to dig into the domain details together with a committed customer and a motivated team. With these key factors and the application of an appropriate combination of well-established as well as some newer methods and tools we were able to efficiently elicit, refine and validate requirements. From this specific context we try to derive implications for innovative requirements analysis. We argue that in projects that go beyond simple, well defined and well understood applications, automated requirements analysis is unlikely to lead to a successful specification of a system.

Keywords: requirements analysis, agile development, use cases, automation systems

1 Domain and project context

The project deals with so-called engine test bed systems that are used, for example, in the automotive industry. Functions of such a system are the parameterization and visualization of its real-world components, such as the engine under test, and the sensors, as well as the measured values. The software has evolved over the last two decades and comprises about 1.5 million lines of code, mainly written in C++ and C. Because one of the goals of the project is to improve the current system's usability, it was named FACE. A major hurdle for the users of the current system is the fact that the domain entities they have in mind (such as engines, dynamometers, measurement and conditioning devices) do not match well with the entities in the user interface. For example, a person who configures a test bed would like to deal with a graphical representation of the test bed as shown in Figure 1, with the parameters and measured values associated with the physical components.

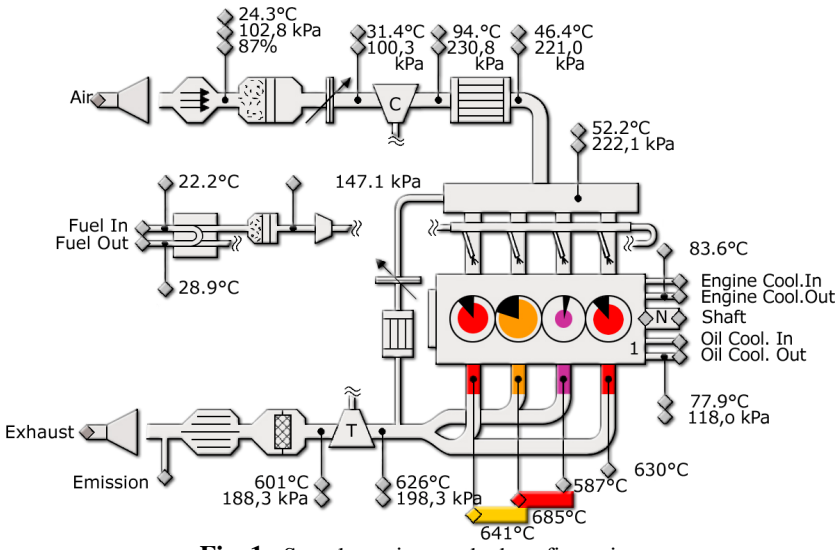
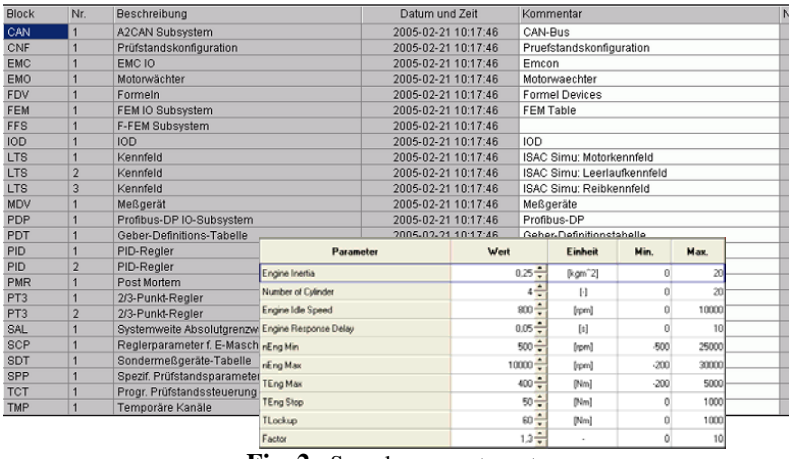


Fig. 1. Sample engine test bed configuration

The current system requires error-prone editing of parameter sets in spreadsheets as depicted in Figure 2 as well as the adaptation of configuration files and scripts scattered in the file system. Using those parameter sets and finding specific parameters is difficult as parameters associated with a physical entity of the test bed are not grouped accordingly. For example, if the engine under test is changed the user has to know by heart which parameters need to be changed. Configuration files and scripts exist in various commercial and proprietary formats and different editors are required for each file type.



Block	Nr.	Beschreibung	Datum und Zeit	Kommentar
CAN	1	A2CAN Subsystem	2005-02-21 10:17:46	CAN-Bus
CNF	1	Prüfstandskonfiguration	2005-02-21 10:17:46	Pruefstandskonfiguration
EMC	1	EMC IO	2005-02-21 10:17:46	Emcon
EMO	1	Motorwächter	2005-02-21 10:17:46	Motorwaechter
FDV	1	Formeln	2005-02-21 10:17:46	Formel Devices
FEM	1	FEM IO Subsystem	2005-02-21 10:17:46	FEM Table
FFS	1	F-FEM Subsystem	2005-02-21 10:17:46	
IOD	1	IOD	2005-02-21 10:17:46	IOD
LTS	1	Kennfeld	2005-02-21 10:17:46	ISAC Simu: Motorkennfeld
LTS	2	Kennfeld	2005-02-21 10:17:46	ISAC Simu: Leerlaufkennfeld
LTS	3	Kennfeld	2005-02-21 10:17:46	ISAC Simu: Reibkennfeld
MDV	1	Meßgerät	2005-02-21 10:17:46	Meßgeräte
PDP	1	Profibus-DP IO-Subsystem	2005-02-21 10:17:46	Profibus-DP
PDT	1	Geber-Definitions-Tabelle	2005-02-21 10:17:46	Geber-Definitionstabelle
PID	1	PID-Regler		
PID	2	PID-Regler		
PMR	1	Post Mortem		
PT3	1	2/3-Punkt-Regler		
PT3	2	2/3-Punkt-Regler		
SAL	1	Systemweite Absolutgrenzw		
SCP	1	Reglerparameter f. E-Masch		
SDT	1	Sondermeßgeräte-Tabelle		
SPP	1	Spezif. Prüfstandsparameter		
TCT	1	Progr. Prüfstandssteuerung		
TMP	1	Temporäre Kanäle		

Parameter	Wert	Einheit	Min.	Max.
Engine Inertia	0,25	[kgm^2]	0	20
Number of Cylinder	4	[-]	0	20
Engine Idle Speed	800	[rpm]	0	10000
Engine Response Delay	0,05	[s]	0	10
nEng Min	500	[rpm]	500	25000
nEng Max	10000	[rpm]	-200	30000
TEng Max	400	[Nm]	-200	5000
TEng Stop	50	[Nm]	0	1000
TLockup	60	[Nm]	0	1000
Factor	1,3	-	0	10

Fig. 2. Sample parameter sets

The system's manufacturer, that is, our customer and research cooperation partner, came up with a fuzzy description of requirements on about 25 pages. The list includes general requirements which are applicable to almost any software system such as maintainability and security issues, and also specific requirements such as the visualization of signal paths. The most important of the overall 19 'requirements' were summarized as follows:

a) Introduce components that support both their visualization and parameterization. The current system consists of parameter sets structured according to software requirements. For example, parameters that have to be loaded at the same point in time are grouped in one parameter set. As a result, parameters are hard to find and using the system requires a lot of knowledge about its implementation because software design decisions are directly reflected in the user interface. The current system does not provide a notion of domain components.
b) Provide different parameter views, including guidance through parameterization tasks. Possible parameter views are the *physical view* only showing parameters describing the hardware; for example, which plugs and cables are used to connect two pieces of hardware and the *functional view* describing aspects such as data type compatibility of connected entities and software functions operating on measured values.
c) Master complexity, for example, by hiding those parameters that are not needed for a specific task. For example, a service task concerned with finding the defect part between the automation system and a certain device does not require knowledge about the simulation model for the device.
d) Compatibility to existing systems, which means supporting a wide range of tools and technologies.
e) Provide only one editor for each class of parameters and eliminate configuration files by using parameter sets instead.
f) Improve performance (start up, export, import, switching between test runs).
g) Do as many checks as possible offline (meaning without being connected to the actual hardware on the test cell), for example, inconsistent measurement and consumption frequencies can be detected offline by comparing parameters of connected components whereas the existence of a piece of hardware in a test bed can only be checked online when the system is connected to the test bed.

The requirements document was accompanied by a confusing diagram that should illustrate the complexity of the current system by showing a set of entities that are potentially relevant for a user and thus need to be displayed. Figure 3 depicts that. You do not need to be able to read and understand it to get an idea of what we mean by confusing diagram.

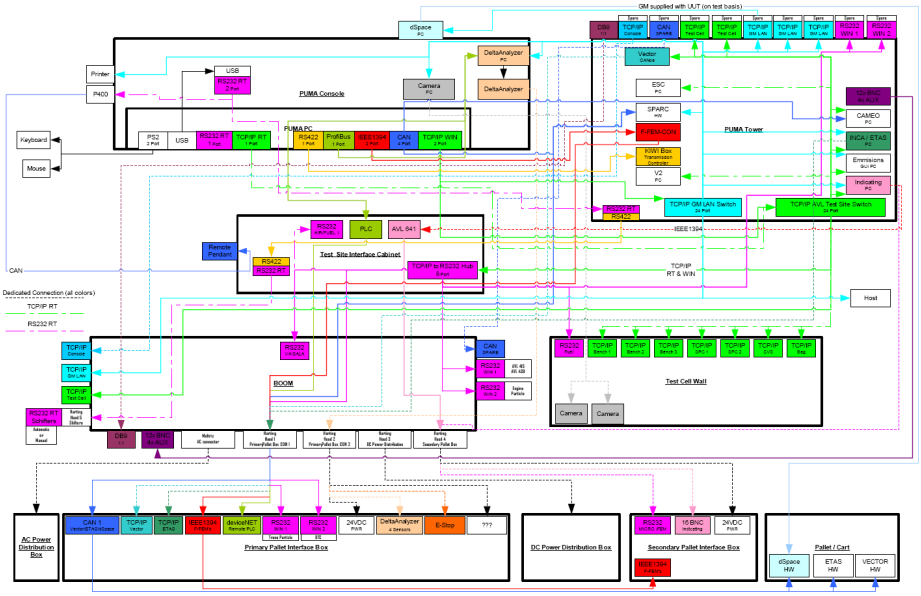


Fig. 3. Diagram as part of the original requirements

Additionally, we received a huge amount (more than one gigabyte) of user documentation, system requirement specifications for the existing system and UML diagrams. The latter consisted of use case diagrams and use cases describing functionality already in specific technical details. Because those documents grew with the software during the last two decades, the documentation was not up to date.

2 The basis of successful requirements analysis

Due to the importance of the FACE project for our customer, the company is fully committed to it and we report to one of its executives. Company representatives with in-depth domain knowledge as well as product managers were available in the requirements analysis phase, which lasted for about 5 months from September 2006 until January 2007. During that phase the team consisted of one top nodge software scientist as team leader, and four young software engineers with little or no project experience. The team leader has extensive software development experience, social skills training, and an additional solid background in automation systems, but had no prior knowledge of the particular engine test bed system. This beginner's mind [12] allowed the team to profoundly analyze the features of the current system as well as its strengths and weaknesses. This is a quality already pointed out by Berry in [1] where he describes a computer-system-savvy person without any knowledge of the domain as the person asking ignorant, not stupid, questions to expose tacit assumptions made by domain-expert stakeholders assuming incorrectly that all other domain-expert stakeholders understand. By making those assumptions explicit, conflicts in the understanding are discovered at an early stage in the software development. The gained domain understanding combined with the team leader's solid background in software science, in particular, component technology, allowed the team to come up with a prototype that was enthusiastically received by the customer's top management in January 2007.

We are convinced that no automated system would have been able to accomplish something close to such a successful requirements analysis and specification based on the available natural language descriptions of the requirements, the current system and its envisioned features.

3 Prototyping-based, agile requirements analysis

The team gained a profound domain understanding in four workshops with customer representatives within the first 6 to 8 weeks months. This process was documented by writing a glossary comprising about 90 terms as well as by analyzing and (re)writing ca. 130 use cases. Then we started to develop a throw-away prototype to show the envisioned system and get rapid feedback from the customer. The prototype applied the following software science concepts to tackle complexity:

- Domain-specific components. The test bed components need to be mirrored in the software system. For that purpose we iteratively defined what we called a Domain Component Description Language (DCDL). Before we could do that we had to become aware of a fundamental misunderstanding when using the term *component*. The customer meant domain-specific components. The development team had components in mind as defined in software science, that is, components describing a unit of composition with contractually specified interfaces and explicit context dependencies only. Such a software component can be deployed independently and is subject to composition by third parties [2]. Though domain components are somehow related to software science components, it was crucial to overcome this misunderstanding. Components for automation systems have to provide multiple views, including a physical, a functional, a parameter, an operations and a configuration view. For the definition of domain components we came up with the DCDL. Domain components allow the customer to manage

a wide variety of configurations that result of different test beds, different devices in each test bed and different test requirements.

- Separation of concerns (SOCs). The term *separation of concerns* can be traced back to [6] and means the splitting of various aspects of software into independent parts that can be dealt with independently. An example where we applied SOCs is in the separation of viewing physical details of a component from viewing its functional details.
- Hierarchical decomposition, collapse & expand. By allowing (domain) components to be built hierarchically, that is, by layering component systems as described in [7], complexity can be managed. Collapse and expand mechanisms help hiding unnecessary details.
- Abstraction, stepwise refinement. Wirth discusses the importance of stepwise refinement in [8]. An example where stepwise refinement is important in the FACE project can be given by the stepwise definition of a test bed. To get an overview of devices in a test bed, it must be possible to create domain components or component groups and to loosely connect them. Only at a later point in time, the physical connections are made and parameters such as plug types and pin assignments are entered into the system.
- Guidance (role-based views, tasks). A useful example, where guidance can improve the work with the system, is problem resolution. Figure 4 shows how we designed a problem resolution view in the early analysis phase.

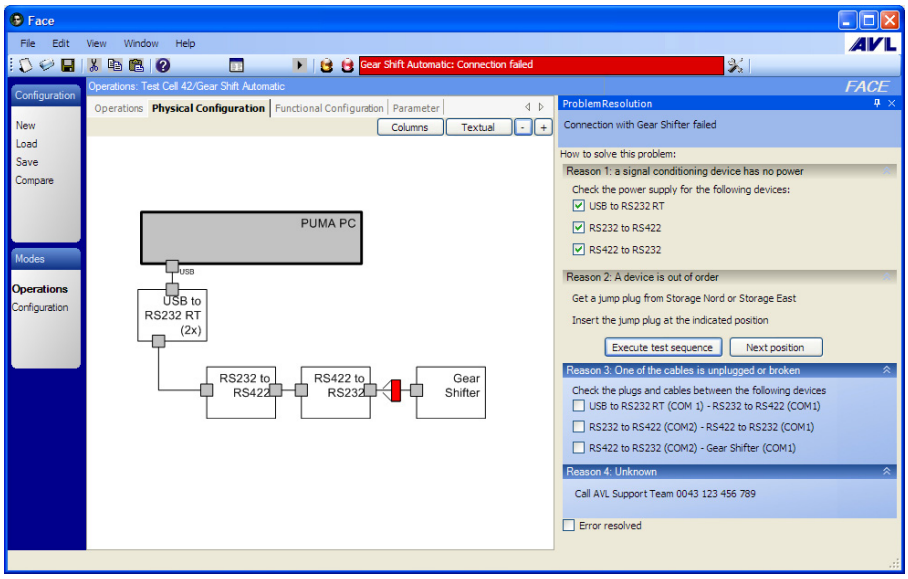


Fig. 4. Prototypical error tracking view

A list of steps that might solve the problem can be generated partly automatically. The user is guided automatically through these steps. Other users of the system, who had similar problems before, might have provided some or all of these steps. A graphical representation could highlight devices in the physical view of the test bed that need to be checked.

- Use of visual representations to manage complexity. Managing complexity by choosing appropriate visual representations of a problem is widely used. Displaying measurement values, parameters and status information of a component can be shown “in place”, that is, close to the graphical representation of a particular domain component (see values and colored regions in Figure 1) and not, as it is the case

in the current system, in some configuration dialogs or error message windows that collect messages from the system as a whole.

We did not start with a prototype right from the beginning because important concepts of the software involved graphical representations that needed to be designed first based on the gradually gained domain understanding. Therefore, numerous concepts and ideas were proposed and discussed in simple drawings on paper, in slide presentations and figures drawn with common drawing tools. Those drawings exemplified how the software could look like for concrete use cases. Figure 5 shows the sketch representing the physical view of a sample test bed configuration. Those pictures were either developed together with the customer or discussed afterwards.

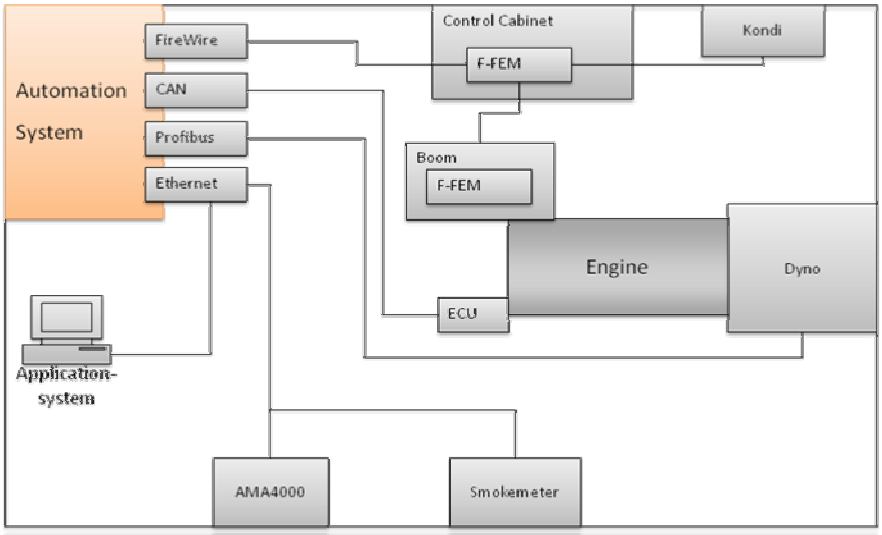


Fig. 5. Physical view of a sample test bed.

Basically, boxes represent domain components which are pieces of hard- or software and are connected to other components. Concepts such as grouping, abstraction by hierarchically structuring components, and different ways of connecting components were applied and refined using these drawings.

As a next step, we analyzed the roles of users who interact with the software. As roles we discerned component engineers, commissioning engineers, test engineers and service engineers. Each role has different goals when using the software and therefore, we refined the use cases and sorted them according to the roles. To get a better understanding of the envisioned system, we started to develop a throw-away prototype. The purpose of this prototype was to show how users interact with the system. For every role, one or two illustrative use cases were chosen and a storyboard was written that describes in detail how the user would operate the software. The prototype implemented these storyboards with pre-drawn pictures according to the user interactions described in the storyboard. This basically means that by using the correct "click-sequence", the correct pictures are shown. Such a prototype can only be operated by someone who knows the storyboards and the click-sequences in detail. To enable the discussion of the prototype with a broader audience, we came up with videos that showed the interaction of the roles with the FACE prototype and we explained the concepts and decisions in the audio track.

From the gained domain understanding and the FACE throw-away prototype that served as a vehicle to document and elicit customer requirements, we derived the user requirements specification, that comprises overall 16 core features. The following list summarizes some of these features:

- a domain component framework
- configuring domain components
- combining domain components in groups or hierarchies
- managing domain components in standard or user-defined libraries
- comparing domain components and finding out similarities
- versioning domain components
- undo and redo mechanisms
- creating and recording macros for often used interaction sequences
- enabling users to provide their own experience such as clues for problem resolution
- managing views and user rights

The implementation phase has started in February 2007. We continue with the close interaction and short feedback cycles that are typical for agile development projects. In particular, we apply test-driven development [13].

4 Limits of automated requirements analysis

This real-world project corroborates, in our point of view, that requirements analysis can barely be automated if the stakeholders do not have a clear understanding about a software system. In this case it was the guts feeling of the customer that the current system could be significantly improved. The customer and its team were somehow trapped in the existing system. Knowing too many details and worrying about significant changes made it virtually impossible to come up with appropriate requirements for an overhauled system. That required creativity that cannot be expected from tools. To quote Deming [14]: “As a good rule, profound knowledge comes from the outside, and by invitation. A system cannot understand itself.”

We briefly checked which tools for automating requirements analysis existing today. Examples for automated requirements analysis tools are the Analyst Real Teams System (ARTS) [10] or the Goal Based Requirements Analysis Tool (GBRAT) [5], Juerjens and Shabalin present in [11] an approach for automated verification of UML models. Some tools take requirements in natural language as an input and use linguistic evaluation for the analysis. A research on linguistic tools can be found in [3]. Many of the mentioned approaches are applied to case studies in domains that are well defined and well understood. That precondition is not true for the FACE project. We assume that none of the tools that automate requirements analysis could lead to a successful completion of the FACE requirements analysis, because the available inputs from the customer are too fuzzy and the terminology is not precise enough. In such a situation the sketched agile requirements analysis with short feedback cycles together with the communication vehicle of a throw-away prototype has turned out to be an appropriate requirements analysis method.

References

1. Berry, D. M.: The Importance of Ignorance in Requirements Engineering, Journal of Systems and Software (1995)
2. Szyperski, C., Pfister, C.: Workshop on Component-Oriented Programming, Summary. In Muehlhaeuser, M. (ed.): Special Issues in Object-Oriented Programming – ECOOP96 Workshop Reader. Heidelberg: Dpunkt Verlag (1997)

3. Luisa, M., Mariangela, F., and Pierluigi, I.: Market research for requirements analysis using linguistic tools. Requir. Eng. 9, 1 (Feb. 2004), 40-56. DOI= http://dx.doi.org/10.1007/s00766-003-0179-8 (2004)
4. Gargantini, A. and Morzenti, A.: Automated deductive requirements analysis of critical systems. ACM Trans. Softw. Eng. Methodol. 10, 3 (Jul. 2001), 255-307. DOI= http://doi.acm.org/10.1145/383876.383877 (2001)
5. Anton, A. I., Liang, E., and Rodenstein, R. A.: A Web-based requirements analysis tool. In Proceedings of the 5th international Workshops on Enabling Technologies: infrastructure For Collaborative Enterprises (WET Ice'96) (June 19 - 21, 1996). WET-ICE. IEEE Computer Society, Washington, DC, 238 (1996)
6. Dijkstra, E. W.: A Discipline of Programming. Prentice-Hall, Englewood Cliffs, New Jersey (1976)
7. Szyperski, C.: Component software and the way ahead. In Foundations of Component-Based Systems, G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, New York, NY, 1-20 (2000)
8. Wirth, N.: Program development by stepwise refinement. Commun. ACM 14, 4 (Apr. 1971), 221-227. DOI= http://doi.acm.org/10.1145/362575.362577 (1971)
9. UML, Unified Modelling Language, http://www.uml.org/.
10. Analyst Real Team System, Goda Software, http://www.analysttool.com/.
11. Juerjens, J., Shabalin, P.: Automated verification of UMLsec models for security requirements. In J.-M. Jézéquel, H. Hußmann, and S. Cook, editors, UML 2004 -- The Unified Modeling Language, volume 2460 of LNCS, pages 412--425. Springer (2004)
12. Suzuki, S.: Zen Mind, Beginner's Mind, Weatherhill (1973)
13. Beck, K.: Test-driven development: By example. Addison-Wesley Publishing (2002)
14. Deming, W. E.: The New Economics for Industry, Government, Education - 2nd Edition. MIT Press. ISBN 0-262-54116-5 (2000)

Logic-based Regulatory Conformance Checking

Nikhil Dinesh, Aravind Joshi, Insup Lee, and Oleg Sokolsky

Department of Computer Science
University of Pennsylvania
Philadelphia, PA - 19104, USA
{nikhild,joshi,lee,sokolsky}@seas.upenn.edu

Abstract. In this paper, we describe an approach to formally assess whether an organization conforms to a body of regulation. Conformance is cast as a model checking question where the regulation is represented in a logic that is evaluated against an abstract model representing the operations of an organization. Regulatory bases are large and complex, and the long term goal of our work is to be able to use natural language processing (NLP) to assist in the translation of regulation to logic.
We argue that the translation of regulation to logic should proceed one sentence at a time. A challenge in taking this approach arises from the fact that sentences is regulation often refer to others. We motivate the need for a formal representation of regulation to accomodate references between statements. We briefy describe a logic in which statements can refer to and reason about others. We then discuss preliminary work on using NLP to assist in the translation of regulatory sentences into logic.

1 Introduction

Regulations, laws and policies that affect many aspects of our lives are represented predominantly as documents in natural language. For example, the Food and Drug Administration's Code of Federal Regulations[1] (FDA CFR) governs the operations of American bloodbanks. The CFR is framed by experts in the field of medicine, and regulates the tests that need to be performed on donations of blood before they are used. In such safety-critical scenarios, it is desirable to assess formally whether an organization (bloodbank) conforms to the regulation (CFR).

Conformance checking is a relatively new problem in requirements engineering, which has been gaining attention in industry and academia [1]. A key difference between regulations and other sources of informal requirements is in determining the source of a requirement. The requirements used to design a system often arise from varied places, such as interviews with customers and discussions with domain experts. This makes the identification of requirements a difficult problem. However, since there are consequences associated with disobeying the law, law-makers spend considerable effort in articulating the requirements (as

[1] http://www.gpoaccess.gov/cfr/index.html

normative sentences). As a result, one can informally associate a requirement with a sentence or discourse.

The challenge in conformance checking is that the task of formalizing the requirements is difficult, due to the large size and complexity of regulations. The long term goal of our work is to use natural language processing (NLP) techniques to aid in the formalization of regulation. From the perspective of using NLP for requirements engineering, this area is epecially interesting due to the availability of large corpora of regulations that can serve as a test-bed for NLP techniques.

We approach the problem of formally determining conformance to regulation as a model-checking question. The regulation is translated to statements in a logic which are evaluated against a model representing the operations of an organization. The result of evaluation is either an affirmative answer to conformance, or a counterexample representing a subset of the operations of the organization and the specific law that is violated. A similar approach is adopted by several systems [1–3].

When a violation is detected, the problem could be in one of three places: (a) the organization's operations, (b) the regulation or (c) the translation of the regulation to the logic. To aid in determining the source of the problem, there needs to be a notion of correspondence between the sentences of regulation in natural language and logic. We attempt to maintain a correspondence by translating regulation to logic one sentence at a time. An added benefit of doing this is to be able to focus our NLP efforts at the sentence level.

In this paper, we discuss two related parts of our approach. The first part deals with the issue of designing a logic into which we can translate regulation one sentence at a time. The main difficulty that we encountered in doing this is the problem of *references to other laws*. A common phenomenon in regulatory texts is for sentences to function as conditions or exceptions to others. This function of sentences makes them dependent on others for their interpretation, and makes the translation to logic difficult. In Section 2, we argue (using examples and lexical occurence statistics) that a logic to represent regulation should provide mechanisms for statements to refer to others, and to make inferences from the sentences referred to.[2] In Section 3, we briefly describe the logic that we use to represent regulation.

In the second part of the paper (Section 4), we turn our attention to the problem of using NLP to assist in the translation of sentences of regulation into logic. Section 5 concludes.

2 The Problem of References to Other Laws

In this section, we argue that a logic to represent regulation should provide a mechanism for sentences to refer to others. The discussion is divided into two parts. In Section 2.1, we discuss examples of the phenomenon that we are

[2] A study in [1] suggests that such references between sentences are common in privacy regulation as well.

interested in and how they may be represented in a logic with no mechanism for sentences to refer to others. We then contrast the distribution of some lexical categories in the CFR with newspaper text, which suggest that references to sentences are an important way of expressing relationships between sentences in regulation (Section 2.2).

2.1 Examples

The examples in this section are shortened versions of sentences from the CFR Section 610.40, which we will use through the course of the paper. Consider the following sentences:

(1) Except as specified in (2), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(2) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

(1) conveys an obligation to test donations of blood or blood component for Hepatitis B, and (2) conveys a permission not to test a donation of source plasma (a blood component) for Hepatitis B. To assess an organization's conformance to (1) and (2), it suffices to check whether "all non-source plasma donations are tested for Hepatitis B". In other words, (1) and (2) imply the following obligation:

(3) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B.

There are a variety of logics in which one can capture the interpretation of (3), as needed for conformance. For example, in first-order logic, one can write $\forall x : (d(x) \wedge \neg sp(x)) \Rightarrow test(x)$, where $d(x)$ is true iff x is a donation, $sp(x)$ is true iff x is a source plasma donation, and $test(x)$ is true iff x is tested for Hepatitis B. Thus, to represent (1) and (2) formally, we inferred that they implied (3) and (3) could be represented more directly in a logic.

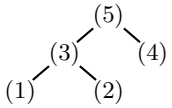
Now suppose we have a sentence that refers to (1):

(4) To test for Hepatitis B, you must use a screening test that the FDA has approved for such use.

The reference is more indirect here, but the interpretation is: "if (1) requires a test, then the test must be performed using an appropriate screening test kit". A bloodbank is not prevented from using a different kind of test for source plasma donations. (4) can be represented by first producing (3), and then inferring that (3) and (4) imply the following:

(5) Every non-source plasma donation must be tested for evidence of infection due to Hepatitis B using a screening test that the FDA has approved for such use.

It is easy to represent the interpretation of (5) directly in a logic. However, (5) has a complex relationship to the sentences from which it was derived, i.e., (1), (2) and (4). The derivation takes the form of a tree:



The examples we have considered are simplified versions of the sentences in the CFR 610.40. In the CFR, (1) has a total of six exceptions, and the exceptions have statements that qualify them further. This process of producing a derived obligation and translating it becomes extremely difficult.

References to other laws are not always hierarchical or acyclic. There are two kinds of circularities that can arise. The first is a syntactic circularity which arises due to vague references. For example, two occurences of the phrase "required testing under this section" can give rise to a cycle if one interprets "this section" as "all the other sentences in this section". However, such phrases typically appear in paragraphs where no tests are required and the cycle can be broken by restricting the references to paragraphs where tests are required. The second kind of circularity is a semantic circularity which can make the regulation paradoxical, e.g., with self referential sentences. Fortunately, we have not observed such circularities.

To summarize, if one wishes to use a logic with no support for referring to other sentences, translating regulation to the logic would involve the following steps: (a) resolving circularites to construct a hierarchy of references, (b) creating derived obligations by moving up the hierarchy, until a set of derived obligations with no references are obtained, and (c) translating the final set of derived obligations to logic.

This procedure would not be problematic if there are few cases of references. In the following section, we discuss the distribution of some lexical categories in the CFR which suggest that this is a very common case. This makes the procedure impractical in terms of the effort that would be involved. The logic that we describe in later sections lets one express references directly, and the resolution of cirularites and creation of derived obligations happen as part of the semantics.

2.2 Distribution of Lexical Categories

In the previous section, we saw several examples of how sentences in regulation refer to others. Natural language offers a variety of devices to relate sentences to others. A large class of such devices fall under the rubric of *anaphora*, which is a means of linking a sentence to the prior discourse. Common examples of such anaphoric items are pronouns and adverbial connectives, e.g., however, instead, furthermore, etc.[3]

Table 1 contrasts the distribution of potentially anaphoric items in the Wall Street Journal (WSJ) corpus, with the CFR. The first three rows show counts

[3] Not all uses of pronouns are anaphoric. Some pronouns are bound by quantifiers, e.g., *every one loves their mother*. We report counts based on occurence of strings and do not distinguinsh between different uses.

Lexical Item	WSJ	CFR
he, she, him, her	8564	297
it, its	15168	2502
they, their	4500	862
ADV1	3162	2402
ADV2	2453	349
such	662	3028
References to other laws	-	18509

Table 1. Differences in the distribution of some anaphoric lexical items in the Wall Street Journal (WSJ) corpus and the CFR. Both the WSJ and the CFR have approximately 1M words.

of pronouns, and the CFR has a markedly lower number of pronouns than the WSJ. The next two rows show counts of adverbial connectives. ADV1 comprises of the connectives *also, however, in addition, otherwise, for example, therefore, previously, later, earlier, until* and *still*. These connectives have specialized uses in the CFR and tend to be quite frequent, with *otherwise* being the most frequent in the CFR (517 cases). ADV2 is a set of 48 adverbial connectives annotated by the Penn Discourse Treebank [4] excluding those in ADV1, e.g., *instead, as a result, nevertheless*.[4] The connectives in ADV2 are significantly more frequent in the WSJ than in the CFR.

The last two rows in Table 1 show two common ways of establishing relationships between sentences in the CFR. The adjective *such* is a common way of refering to a set discussed in an immediately preceding law, e.g., *such tests*. The last row counts explicit references to other law, by searching for phrases like *this section*, or references to section and paragraph identifiers. Of the categories we considered this is by far the most frequent in the CFR.

3 A Logic that Allows References Between Laws

In this section, we describe the logic that we attempt to translate the regulation into. The description in this section is brief and informal, and introduces only the machinery needed to clarify the discussion in Section 4. The logic builds on ideas from Reiter's default logic [5] and Kripke's truth predicate [6]. Consider our examples again:

(6) Except as specified in (7), every donation of blood or blood component must be tested for evidence of infection due to Hepatitis B.

(7) You are not required to test donations of source plasma for evidence of infection due to Hepatitis B.

(6) and (7) are represented as follows:

[4] The Penn Discourse Treebank annotates discourse connectives and their arguments. The current version of the corpus is available at http://www.seas.upenn.edu/~pdtb

- 6.**o**: $d(x) \wedge \neg \text{by}_7(\neg\Diamond test(x)) \rightsquigarrow \Diamond test(x)$ and
- 7.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$

First, consider the formula 7.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$. This is read as "It is permitted that if y is a donation of source plasma, then it is not tested eventually". The letter **p** denotes permission, $d(y)$ asserts that y is a donation, $sp(y)$ asserts that y consists of source plasma, $test(y)$ asserts that y is tested, and $\Diamond$is the linear temporal logic (LTL) operator eventually. The connective $\rightsquigarrow$ is a variant of implication which we will discuss in what follows.

Now consider the subformula $\text{by}_7(\neg\Diamond test(x))$. This is read as "By the law (7), x is not tested eventually". We note that this subformula should hold iff y is a donation of source plasma. And finally, 6.**o**: $d(x) \wedge \neg \text{by}_7(\neg\Diamond test(x)) \rightsquigarrow \Diamond test(x)$ can be paraphrased as "It is obligated that if x is a donation and it is not the case (7) permits that x is not tested eventually, then x must be tested eventually". The letter **o** denotes obligation.

Formulas in the logic are evaluated with respect to sequences of states of an implementation (in a manner similar to LTL). Each state is associated with a set of objects and a way of evaluating predicates. Consider a state with an object o_1 such that $d(o_1)$ and $sp(o_1)$ hold, i.e., o_1 is a donation of source plasma. Assume that we have a sequence of states where o_1 is not tested.

We first evaluate 7.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$ with respect to all variable assignments. When y is assigned the value o_1, the precondition $d(y) \wedge sp(y)$ is true, and we *annotate* the state with $\neg\Diamond test(o_1)$. This annotation happens regardless of whether $\neg\Diamond test(y)$ is true or false under the variable assignment.

Next, we evaluate 6.**o**: $d(x) \wedge \neg \text{by}_7(\neg\Diamond test(x)) \rightsquigarrow \Diamond test(x)$. When x is assigned the value o_1, $d(x)$ is true. To evaluate $\text{by}_7(\neg\Diamond test(x))$ we check if there is an annotation (ψ) on the state such that $\psi \Rightarrow \neg\Diamond test(o_1)$ is valid, i.e., a theorem in LTL. Since $\neg\Diamond test(o_1)$ is an annotation, it is an appropriate candidate for ψ and we conclude that $\text{by}_7(\neg\Diamond test(x))$ is true. Hence the precondition $d(x) \wedge \neg \text{by}_7(\neg\Diamond test(x))$ is false, and the obligation is vacuously satisfied.

When considering a non-source plasma donation (o_2), no annotation is provided by 7.**p**: $d(y) \wedge sp(y) \rightsquigarrow \neg\Diamond test(y)$. We will not be able to find ψ such that $\psi \Rightarrow \neg\Diamond test(o_2)$ is valid. This will make the precondition $d(x) \wedge \neg \text{by}_7(\neg\Diamond test(x))$ true, and if a test is not performed eventually, a violation will be detected. Violations are detected only with respect to obligations. Permissons do not produce violations and are relevant to conformance only via references from an obligation.

4 NLP as an Aid in Formalizing Regulation

In this section, we discuss preliminary work on using natural language processing (NLP) to aid in creating a logic-based representation of regulation. We emphasize that we use NLP purely as an assistive technology and do not attempt to replace the human user.[5]

[5] The intended users of our system are designers of software in the organization being regulated.

We approach the problem using the annotation-learning methodology, which has been used for a variety of tasks in NLP, e.g., parsing, computing predicate-argument structure, named-entity recognition etc. The annotation-learning methodology proceeds as follows:

1. Define a representation (logic) to be computed from the text
2. Manually describe/annotate how units of text correspond to units of the desired representation. The number of examples manually annotated depends on the needs of the application.
3. Train a (statistical) learning algorithm to compute the representation.

Our focus upto this point has been on Steps 1 and 2, i.e., designing the logic and formulating an annotation scheme to associate natural language and logic. In order for the methodology to be successful, it should be possible for a human to describe how she went from natural language to logic. Such a description would take the form of an annotation guideline. For example, [7] gives guidelines for annotating phrase structure on sentences, and [8] gives guidelines for annotating discourse relations. The process of formulating guidelines is typically one of iterative refinement. We begin by fixing a representation, and then annotating a few sentences with this representation. The problematic cases are analysed, resulting in revisions to the guidelines, and the process repeats.

We have made three annotation passes over 100 sentences and are in the process of refining guidelines. The rest of this section describes what we are attempting to annotate and the difficulties encountered. In Section 4.1, we decompose the annotation process into three steps. Some of the key problems encountered are discussed in Sections 4.2 and 4.3. Section 4.4 discusses related work. A preliminary discussion of our approach appears in [9]. We have since extended the logic and annotation guidelines.

4.1 Annotating regulatory sentences with logic

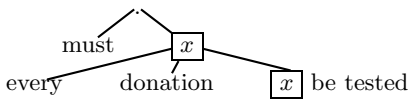
Many logics are semantically adequate for the application of conformance checking. However, to be able to describe or annotate how a statement in logic is obtained from natural language, the logic and natural language need to be syntactically isomorphic. (8) and (9) are examples of what we mean by syntactically isomorphic statements in natural language and logic.

(8) Every donation must be tested

(9) 8.**o**: $donation(x) \leadsto tested(x)$

We note that predicates such as $tested(x)$ need to be refined to accomodate references between laws, and we discuss this issue in what follows. We now sketch the procedure for associating (8) and (9).

First, (8) is mapped to the *abstract syntax tree* (AST):



The AST is obtained from (8) by moving the modal *must*, followed by moving the phrase *every donation* to the front of the sentence. While moving a word or phrase, a variable is optionally left behind as a placeholder.

The second step is to associate leaf nodes which are not the leftmost child of their parent with components of the formula. *donation* is associated with the predicate $donation(x)$, and *x must be tested* is associated with the predicate $tested(x)$.

Given the associations for non-leftmost leaves, the leftmost leaves are associated with operations that combine the associations of their siblings to create an association for the parent. *every* is associated with an operation that combines the associations of its siblings to associate $donation(x) \leadsto tested(x)$ with its parent. Finally, *must* is associated with an operation that takes $donation(x) \leadsto tested(x)$ and associates (9) with its parent. This procedure for associating natural language to logic can be broken into three steps:

1. Converting a sentence to an AST
2. Associating the non-leftmost leaves of the AST with components of the logic
3. Associating the leftmost leaves with combination operations

This decomposition of the problem is the one adopted (modulo terminology) in theoretical linguistics [10]. Of these steps, our goal is to achieve automation with a good level of accuracy for Step 1. For Steps 2 and 3, we can only envision partial automation in the immediate future. The goal is to design appropriate interfaces to assist the user in performing these steps. We now discuss some of the challenges in associating regulatory sentences with ASTs (Section 4.2). We then turn to a discussion of some issues related to Steps 2 and 3 in Section 4.3.

4.2 Annotating sentences with ASTs

The AST produced from a sentence is a resolution of scope ambiguities. The sentence (8) above is simple in comparison to the sentences that one encounters in regulatory text, where a sentence has multiple noun phrases and modalities. Consider the following sentence from CFR 610.1 (the AST is shown in Figure 1):

(10) No lot of any licensed product shall be released by the manufacturer prior to the completion of tests for conformity with standards applicable to such product.

(11) 10.**o**: $licensedProduct(x) \wedge lotOf(y,x) \wedge priorTo(\varphi) \wedge manufacturer(z) \leadsto \neg releasedBy(y,z)$

For simplicity, we omit some details from (11). The phrase *the completion of tests for conformity with standards applicable to such product* involves a reference to other laws, i.e., the applicable standards appear in various places in Part 610. The subformula $priorTo(\varphi)$ in (11) can be formalized using a variant of the technique discussed in Section 3.

We now discuss some issues related to (10), (11) and the AST in Figure 1.

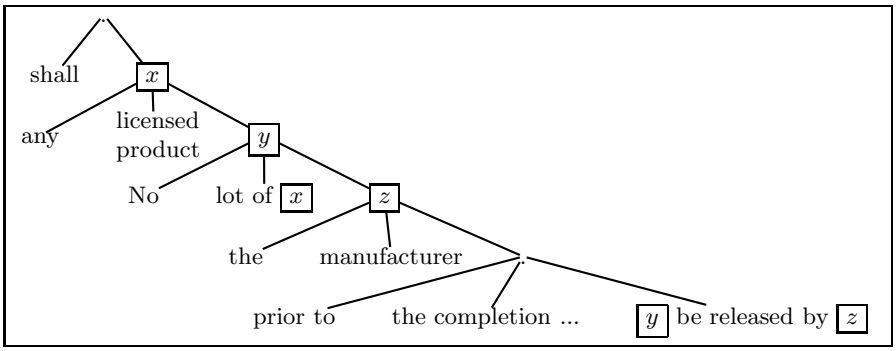


Fig. 1. AST for (10). The structure for the noun phrase *the completion of tests for conformity with standards applicable to such product* is not shown.

Consider again the phrase *the completion of tests for conformity with standards applicable to such product*. While we can give this phrase an internal stucture in the AST, we do not know how to associate it structurally to its formal interpretation. In other words, from the perspective of translation, the phrase has to be treated as an idiom of sorts. In annotating a sentence with its AST, we give such phrases an internal structure and leave the problem of treating it as an idiom to subsequent steps.

Another issue is the question of what to move. In many linguistic theories, only quantificational noun phrases, e.g., *any product*, are treated as candidates for movement. In our annotation scheme, the constructs that are moved are noun phrases, coordinated and subordinated phrases/clauses, relative clauses, and some modals and adverbs. This lets use describe the scopal interaction of these constructions without having to construct a separate phrase structure tree, thus saving annotation effort.

With subordinating conjunctions, modals and adverbs, we move only those constructions that express conditional, temporal or deontic relations. For example, a modality such as *deemed necessary* would not be moved, as we do not know how to formalize this as an operator. This gives rise to some difficult cases where the scope of an unformalized construction interacts with another which is moved:

(12) You must perform one or more such tests, as necessary, to reduce adequately and appropriately the risk of transmission of communciable disease.

In (12) it is unclear how to order *one or more such tests*, *as necessary* and *to reduce adequately and appropriately the risk of transmission of communciable disease*. The intended interpretation of this sentence (which is also unclear) is "if a test is required, then it must be performed repeatedly until a conclusive

result is obtained".[6] In such cases, we construct an AST following the surface order of the phrases.

The point to take from this discussion is that not all the structure provided in an AST can be mapped directly to logic. On occasion one has to "undo" some of the movements in order to perform the association. An analogous situation arises in the problem of alignment in machine translation (between natural languages). One cannot always find a syntactically isomorphic translation of an English sentence into French. Certain constructions have to be treated idiomatically. In translating to logic, the number of constructions that we have to treat idiomatically give us a way to evaluate the syntactic expressive power of the logic. If there are many such constructions, it would suggest that the logic needs to be extended. We now discuss issues in associating the leaves of the AST with formulas in logic.

4.3 Associating the leaves of ASTs with logic

In Section 4.1, we gave the AST for the sentence "every donation must be tested". The leaf node "x be tested" was associated with the predicate $tested(x)$. In order to accomodate references between laws, we need to be able to infer, for example, that "if no tests are required, then a test for Hepatitis B is not required". Such inferences would not succeed if we used predicates such as $tested(x)$ and $testedForHepatitisB(x)$ (we need $\neg tested(x) \Rightarrow \neg testedForHepatitisB(x)$ to be valid). To handle such cases, we approach the definition of the set of predicates in two steps, which we describe below.

The first step is creating a schema. A schema is a set of class definitions. A class definition consists of a set of attribute definitions, and an attribute definition is a name associated with a type. The types of attributes are taken to be either atomic values (numbers or strings), references or sets of references. For example, the class *Donation* has an attribute named *tests* which is a set of references to objects in the class *Test*.

Predicates are treated as assertions over instances of the schema and are defined using a description logic. The logic that we use combines graded modal logic (modal logic with counting quantifiers), and hybrid logic (which allows one to refer to particular objects). The predicate "$testedForHepatitisB(x)$" is formalized as: $@_x(\exists tests : (purpose = \text{Hepatitis B}))$, read as "at the object referred to by x, there is an object referenced in the *tests* attribute and the *purpose* attribute of the test object has the value "Hepatitis B".

Given a set of documents, there are many ways one could create a schema, depending on domain knowledge and taste. Designing NLP-based interfaces to aid in the extraction of schemas has been explored in the past [11, 12]. Our goal is to adapt previous work to the regulatory domain. Both creating the schema and defining the predicates will require significant manual intervention.

The key challenge in translating natural language to logic (for our application) is being able to decompose the problem into steps that depend mostly on

[6] The intended interpretation was clarified in a memo released by the FDA.

the text, and steps that depend mostly on domain knowledge. We believe that the computation of ASTs and the creation of schemas can be tied closely to the text, and as more documents are formalized better accuracy can be achieved. The problem of creating predicate definitions would benefit from further decomposition, and to our knowledge, this is an open problem.

4.4 Related Work

The problem of translating natural language to logic has received much attention in theoretical linguistics [13]. There are many problems both in the design of logic and the translation procedure that have yet to be resolved. More recently, there have been efforts in NLP to automate this translation for some applications.

[14, 15] show that a good degree of automation can be achieved when the text is constrained. The sentences considered are queries to geographical database, e.g., "Which states does the river Delaware run through?". The specific corpus considered associates each sentence to logic. The associations between components of a sentence and logic are computed during the learning phase. While this approach reduces the annotation effort, the inference of associations during the learning step becomes more difficult.

[16] describes a corpus annotated using a manually crafted Head-driven Phrase Structure Grammar (HPSG). In addition to parse trees, a translation to logic is associated. The logic produced is similar in spirit to the ASTs that we annotate. We do not adopt this approach directly for two reasons. First, the annotation of ASTs avoids the overhead of creating phrase structure trees. Second, the logic produced in [16] introduces a large number of predicates (approximately one per word), and this makes the formulas large and difficult to refine. The leaves of the AST are typically phrases, and we have found in case studies that it is easier to define predicates at this level of granularity.

[17] discusses an approach to computing wide-coverage semantic interpretation. The goal is to be able to produce approximate translations in first-order logic and carry out inferences. Similar problems arise in the definition of predicates. The envisioned applications are those for which some errors in the logic produced are tolerable. For our application, while it may be impossible to avoid errors, the goal is to provide a correct translation of a sentence. This involves a careful analysis of modalities, which is not possible in current wide-coverage techniques.

5 Conclusions and Future Work

We have motivated the need for a formal representation of regulation to accomodate references between laws (Section 2). We described, in Section 3, a logic that accomodates certain kinds of references, i.e., those appearing in preconditions. There is also the need for reference in postconditions, to express naturally cases where one law cancels obligations and permissions given by another. We are currently working on extending the logic to allow such references.

In Section 4, we described preliminary work on using NLP to assist in creating the formal representation of regulation. In NLP, the focus has been on computing information tied to the surface structure of the sentence, such as parse trees and predicate-argument structure. However, in formalizing requirements, we are often interested in inferences drawn from sentences and the context. Relating these inferences back to the surface structure of a sentence poses interesting challenges to both NLP and formal methods.

References

1. Breaux, T.D., Vail, M.W., Anton, A.I.: Towards regulatory compliance: Extracting rights and obligations to align requirements with regulations. In: Proceedings of the 14th IEEE International Requirements Engineering Conference. (2006)
2. Abrahams, A.: Developing and Executing Electronic Commerce Applications with Occurrences. PhD thesis, Univeristy of Cambridge (2002)
3. Giblin, C., Liu, A., Muller, S., Pfitzmann, B., Zhou, X.: Regulations Expressed as Logical Models (REALM). In Moens, M.F., Spyns, P., eds.: Legal Knowledge and Information Systems. (2005)
4. Miltsakaki, E., Prasad, R., Joshi, A., B.Webber: The Penn Discourse Treebank. In: LREC. (2004)
5. Reiter, R.: A logic for default reasoning. In: Readings in nonmonotonic reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1987) 68–93
6. Kripke, S.: Outline of a theory of truth. Journal of Philosophy **72** (1975) 690–716
7. Bies, A., Ferguson, M., Katz, K., MacIntyre, R.: Bracketing guidelines for Treebank II style Penn Treebank Project. ftp://ftp.cis.upenn.edu/pub/treebank/doc/manual/root.ps.gz (1995)
8. The PDTB Group: The Penn Discourse Treebank 1.0 Annotation Manual. Technical Report IRCS-06-01, IRCS (2006)
9. Dinesh, N., Joshi, A.K., Lee, I., Webber, B.: Extracting formal specifications from natural language regulatory documents. In: Proceedings of the Fifth International Workshop on Inference in Computational Semantics. (2006)
10. May, R.: Logical Form: Its structure and derivation. MIT Press (1985)
11. Overmeyer, S.P., Lavoie, B., Rambow, O.: Conceptual modeling through linguistic analysis using lida. In: 23rd International conference on Software Engineering. (2001) 401–410
12. Bryant, B.R.: Object-oriented natural language requirements specification. In: ACSC 2000, The 23rd Australasian Computer Science Conference. (Jan 2000)
13. Heim, I., Kratzer, A.: Semantics in Generative Grammar. Blackwell (1998)
14. Zettlemoyer, L.S., Collins, M.: Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In: Proceedings of UAI. (2005)
15. Wong, Y.W., Mooney, R.J.: Learning synchronous grammars for semantic parsing with lambda calculus. In: Proceedings of ACL. (2007)
16. Oepen, S., Flickinger, D., Toutanova, K., Manning, C.: LinGO Redwoods: A rich dynamic treebank for HPSG. In: Proceedings of the workshop on treebanks and linguistic theories. (2002)
17. Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representations from a CCG parser. In: Proceedings of COLING. (2004)

MOVING BEYOND SENTENCE: PENN DISCOURSE TREEBANK

Aravind K. Joshi
Institute for Research in Cognitive Science
University of Pennsylvania 3401 Walnut Street, Suite 400A
Philadelphia, PA 19104-6228

Abstract

In the area of Natural Language Processing (NLP) parsing technology has advanced significantly. The existence of large scale annotated corpora (at the sentence level), for example, the Penn Treebank (PTB), has played a key role in the development of NLP techniques such as parsing, at the sentence level. I will briefly describe the Penn Discourse Treebank (PDTB)^* , a corpus in which we annotate the discourse connectives (explicit and implicit) and their arguments together with "attributions" of the arguments and the relations denoted by the connectives, and also the senses of the connectives. Discourse connectives are like higher level predicates taking clauses as their arguments and thus serve as bridges from the sentence level to the discourse level. The original motivation for undertaking the development of PDTB was to provide a resource for lifting sentence level processing to the discourse level. However, now that this resource has been built, several possible applications are suggested which might be achievable without full discourse parsing. From this perspective, I believe the PDTB resource will be of interest to the participants of the Monterey Workshop, 2007^** . * This 1 million-word corpus (same as the WSJ corpus used by the Penn Treebank (PTB) for syntactic annotation) is expected to be released later this year. ** A preliminary effort along these lines has been suggested in Dinesh et al. (2007).

On the Identification of Goals in Stakeholders Dialogs

Leonid Kof

Fakultät für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei München, Germany
kof@informatik.tu-muenchen.de

Abstract. Contradictions in requirements are inevitable in early project stages. To resolve these contradictions, it is necessary to know the rationales (goals) that lead to the particular requirements. In early project stages one stakeholder rarely knows the goals of the others. Sometimes the stakeholders cannot explicitly state even their own goals. Thus, the goals have to be elaborated in the process of requirements elicitation and negotiation.
This paper shows how the goals can be guessed by systematic analysis of stakeholders dialogs. The guessed goals have to be presented to the stakeholders for validation. Then, when the goals are explicitly stated and validated, it becomes easier to resolve requirements contradictions.

1 Introduction

1.1 Goal-Oriented Requirements Engineering

A *goal* in requirements engineering is "an objective the system under consideration should achieve" [1]. Goals build the basis for requirements elicitation process, as they tend to be more stable than single requirements: For example, the goal "air traffic security should be improved" is less likely to change during the project than the requirement "screening procedure X should be applied from now on."

In early project stages it is normal that the goals or requirements of different stakeholders contradict to each other. This makes it even more important, to identify the goals as early as possible: A conflict in requirements may result from the peculiarities of the intended solutions, whereas a goal conflict is much more fundamental.

1.2 Case Study: Airport Security Screening

The procedure for goal identification, presented in this paper, is evaluated on a small case study on an airport screening system. The case study is just a two-page document, representing an online stakeholder discussion [2]. This document does not contain any explicitly stated requirements.

There are three stakeholders participating in the discussion: a representative of the Transportation Security Administration, a representative of the Federal Aviation Administration, and a representative of the airport screening and security staff. They all agree on the goal that the air traffic security should be improved, but they see different problems and propose different solutions to the common goal. On the total, they write just 4-5 paragraphs each, which is surely not enough to identify all requirements. However, their goals become apparent even in these short statements.

1.3 Outline

The remainder of the paper is organized as follows: Section 2 presents the goals identified as the result of ad-hoc analysis of the case study document. Section 3 shows how the goal identification can be systematized, and, finally, Section 4 summarizes the whole paper.

2 Goals are not Always Explicit: Ad-Hoc Identification of the Goals in the Case Study

In the ideal world every stakeholder could explicitly state her goals and identify contradictions to other stakeholders' goals. The small case study, treated in this paper, shows that this is not the case in the real world. In the stakeholders dialog the goals are mostly implicit, they manifest themselves in proposals that a stakeholder makes and in objections to the proposals made by others. For example, in the case study the FAA officer opens the discussion with the statement that "We have to ban on airplane passengers taking liquids on board *in order to increase security following the recent foiled United Kingdom terrorist plot*." In this case the goal is explicitly stated, introduced by the phrase "in order to". The reaction to this statement shows the goal of the airport screening staff, rather indirectly: "Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Channel No. 5 you get False Alarms... *so it is not quite ready for deployment*!" The actual goal is the applicability of the screening techniques in day-to-day operation, not the problem of distinguishing squeeze cheese from explosives.

In the case study we can identify the goals by permanently asking the question, why a certain statement was made by a discussion participant. In this way we can identify the following goals of the stakeholders:

- Goals of the Federal Aviation Administration:
  - improvement of security: "We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot"
  - effectiveness: "We are trying to federalize checkpoints and to bring in more manpower and technology"
- Goals of the Transportation Security Administration:
  - improvement of security
    - pro-active thinking: "We have yet to take a significant pro-active step in preventing another attack everything to this point has been reactive"
    - consistency in regulations: "I think that enforcing consistency in our regulations and especially in their application will be a good thing to do"
- Goals of the airport screening and security staff:
  - applicability of the rules in everyday operation: "Technologies that could help might work well in a lab, . . . , so it is not quite ready for deployment", "It's not easy to move 2 million passengers through U.S. airports daily"
  - cost effectiveness for the airlines: "I mean an economic threat is also a threat"

- consistency in rules: “There are constant changes in screening rules - liquids/no liquids/3-1-1 rule”

These goals are not contradiction-free. By analyzing the document it is possible to identify following contradictions:

– proactive thinking, which is a TSA goal, vs. cost effectiveness, which is an FAA goal. Actually, this is not necessarily a contradiction, but it sounds like a contradiction in the dialog.
– responsibility for the security checks: airlines become responsible, which is an FAA goal, vs. the authority currently performing the checks remains responsible.
– acceptability of false positives: acceptable for FAA, not acceptable for the screening staff

Probably due to the fact that each stakeholder considers his own goals as obvious, no one ever explicitly states them. Instead, each stakeholder presents solutions that seem adequate to him and explains why he thinks the solutions proposed by others are problematic. This observation about indirect goal statements will be used in the next section in order to systematize and potentially automate the identification of goals.

3 Case Study, Systematization of Goal Identification

In the previous section we identified the goals by close inspection of the text. Now we want to systematize the inspection procedure. To systematize the analysis, we apply two observations to every paragraph:

– Phrases like “have to”, “in order to”, or similar, directly show the goal. The negation of such a phrase shows the current state of affairs, that should be improved.
– If the first sentence of the paragraph does not contain any of the above phrases, it states the reason why the previous paragraph is problematic. In this case, the negation of this sentence shows the stakeholder’s goal.

3.1 Evaluation of the rule application

Table 1 shows the results of the application of the above rules to the case study. The application was performed manually by adhering to the rules as strictly as possible. This means that in some cases not the first sentence of the paragraph but the first meaningful one was taken into consideration. For example, statements like “come on”, “well…”, “we can deal with it” were ignored, as they do not contribute to the identification of the goals. For this reason Table 1 sometimes lists other than the first sentence of the paragraph.

It is important to emphasize that the negations listed in Table 1 were not constructed by purely syntactical deletion or adjoining of “not” at some position in the sentence. Such syntactical negations had to be generalized. For example, “It’s not easy to move 2 million passengers…”, statement from paragraph 4, was negated to “It should be easy to move 2 million passengers…” and then generalized to “The screening system has to

	Sentence	**State of the art/Goal**	**Evaluation**
1	We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot.	State of the art: we do not ban passengers taking liquids, terrorist plot like in the UK is possible. Goals: ban passengers taking liquids, increase security	
2	Technologies that could help might work well in a lab, but when you use it dozens of times daily screening everything from squeeze cheese to Channel No. 5 you get False Alarms ...	Goals: technologies should work not only in the lab, without false alarms	Goal correctly identified
3	Generating false positives helped us stay alive; maybe that wasn't a lion that your ancestor saw, but it was better to be safe than sorry.	No goal identifiable	—
4	It's not easy to move 2 million passengers through U.S. airports daily.	Goal: the screening system has to handle 2 million passengers daily	Goal correctly identified
5	We can deal with it. What if you guys take frequent breaks?	No goal identifiable	—
6	Sounds good though we do take breaks and are getting inspected.	No goal identifiable	—
7	We have yet to take a significant pro-active step in preventing another attack everything to this point has been reactive.	State of the art: We do not take pro-active steps. Goal: We have yet to take pro-active steps	Goal correctly identified
8	On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect.	Goal: we should not spend too much on the screening procedure, it should remain affordable	Goal correctly identified
9	We need to think ahead. For instance, nobody needs a metal object to bring down an airliner, not even explosives.	Goal: identify other types of objects to be banned	Goal correctly identified
10	Airlines need to take the lead on aviation security.	Goal: Airlines need to take the lead on aviation security, not FAA.	FAA Goal correctly identified
11	Sir, a lot of airlines are not doing well and are on the Government assistance.	Goal: Airlines should not be responsible for additional cost-intensive tasks.	Goal correctly identified
12	I think that enforcing consistency in our regulations and especially in their application will be a good thing to do.	State of the art: regulations are inconsistent Goal: regulations should be consistent.	Goal correctly identified
13	Ok, we had very productive discussion	No goal identifiable	—

Table 1. Application of the hypothesis to the case study

handle 2 million passengers daily". In a similar way, "On each dollar that a potential attacker spends on his plot we had to spend $1000 to protect" was negated to "On each dollar that a potential attacker spends on his plot we should not spend $1000 to protect" and generalized to "The screening procedure should remain affordable".

It is easy to see that Table 1 contains all the goals identified by ad-hoc analysis in Section 2. However, it is necessary to bear in mind that the case study was rather small and that both analysis runs, ad-hoc and systematic, were performed by the same person, which makes the results potentially biased. Thus, to properly evaluate the rules for goal identification, a controlled experiment or computer implementation of the procedure is necessary.

3.2 Possible Implementation

To implement the introduced procedure for goal identification, it is necessary to solve two problems:

- It is necessary to define what a meaningful sentence is, in order to analyze the first meaningful sentence of every paragraph.
- Negation is not always possible by simple deletion or adjoining of "not". Furthermore, negated sentences have to be generalized.

The first problem is relatively simple from the point of view of computational linguistics: We could eliminate sentences without grammatical subject, like "come on" and "well…", as well as questions, like "What do you suggest?" in the case study document. This would work for most paragraphs of the considered case study.

The second problem, the negation, is much more difficult. To cope with the grammatical negation, we can try to translate every sentence to discourse representation structure (DRS) [3, 4]. DRS can be translated to first-order logic, thus, when performing negation on the DRS level, we would obtain a logical negation. On the DRS level we could negate different pieces of the DRS, which would correspond to the negation of different clauses of the sentence. Then, we would have to present different negations to the user in order that she selects the correct one. In this way we can get, for example, from "On each dollar that a potential attacker spends on his plot we *had to* spend $1000 to protect" to "On each dollar that a potential attacker spends on his plot we *should not* spend $1000 to protect". However, even when we have the grammatical negation, we have to generalize it. For example, in the case study we had to generalize "On each dollar that a potential attacker spends on his plot we *should not* spend $1000 to protect" to "The screening procedure should remain affordable". This is impossible without profound knowledge of the world, so-called common sense.

4 Summary

In this paper a method for identification of stakeholders' goals by analyzing stakeholders' dialogs was introduced. This method bases on two key assumptions:

- Sentences containing certain keywords directly represent the goal.

– Otherwise, if the sentence is the first meaningful sentence of the paragraph, its negation represents the goal.

This strategy for goal identification is very similar to the strategy proposed by van Lamsweerde [5], consisting of three rules:

– Sentences containing certain keywords directly represent the goal.
– Asking the "why" question for already identified goals helps to identify more abstract goals.
– Asking the "how" question for already identified goals helps to identify more concrete goals.

The second rule used in this paper, the negation rule, can be seen as an application of the why-rule to the dialog: We are just asking the question, why a particular statement was made. One of the reasons to start a new dialog segment is stakeholder's disagreement with the previous proposal. In this case the negation of the first statement shows the reason for the disagreement, which is some goal of the stakeholder.

Explicit goal identification is important for several reasons. Goals serve to achieve requirements completeness and pertinence, managing requirements conflicts, etc [1]. The presented approach is especially suitable to manage requirements conflicts when negotiating requirements: In the Win-Win negotiation approach [6] requirements conflicts are resolved in such a way that the *goals* of every stakeholder remain satisfied. In the case of goal conflicts such a resolution is impossible. Thus, identification of goals and goal conflicts, as in the presented paper, contributes to the identification of potential problems early in the development process.

References

1. van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering, IEEE Computer Society (2001) 249–263
2. Case Study: Air Traveling Requirements Updated (Blog scenario): (2007) `http://fabrice.kordon.free.fr/Monterey2007/invitation_files/case-1.pdf`, accessed 06.04.2007.
3. Bos, J., Clark, S., Steedman, M., Curran, J.R., Hockenmaier, J.: Wide-coverage semantic representations from a CCG parser. In: COLING '04: Proceedings of the 20th international conference on Computational Linguistics, Morristown, NJ, USA, Association for Computational Linguistics (2004) 1240
4. Bos, J.: Towards wide-coverage semantic interpretation. In: Proceedings of the 6th International Workshop on Computational Semantics (IWCS 6). (2005) 42–53
5. van Lamsweerde, A.: Requirements engineering in the year 00: a research perspective. In: ICSE '00: Proceedings of the 22nd international conference on Software engineering, New York, NY, USA, ACM Press (2000) 5–19
6. Grünbacher, P., Boehm, B.W., Briggs, R.O.: EasyWinWin: A groupware-supported methodology for requirements negotiation (2002) `http://sunset.usc.edu/research/WINWIN/EasyWinWin/index.html`, accessed 06.04.2007.

Profiling and Tracing Stakeholder Needs

Pete Sawyer, Ricardo Gacitua, Andrew Stone

Lancaster University, Lancaster, UK. LA1 4WA
{sawyer, gacitur1}@comp.lancs.ac.uk, a.stone1@lancs.ac.uk

Abstract. The first stage in transitioning from stakeholders' needs to formal designs is the synthesis of user requirements from information elicited from the stakeholders. In this paper we show how shallow natural language techniques can be used to assist analysis of the elicited information and so inform the synthesis of the user requirements. We also show how related techniques can be used for the subsequent management of requirements and even help detect the absence of requirements' motivation by identifying unprovenanced requirements.

1 Introduction

Since "the majority of requirements are given in natural language, either written or orally expressed" [1] the application of natural language processing (NLP) to Requirements Engineering (RE) has been investigated by many researchers. In this paper we discuss the use of *shallow* NLP techniques in the early stages of transitioning from stakeholders' need to formal designs; the synthesis of user requirements that are informed by information elicited from the stakeholders and the subsequent management of this information. We also consider the conundrum posed by missing or suppressed information and the perhaps paradoxical potential for shallow techniques to detect the absence of information.

2 Assisting the Synthesis of User Requirements

Among the most challenging applications of NLP in RE have been problems where the language used is uncontrolled [2]. Uncontrolled language is characteristic of early-phase RE [3] where the stakeholders not only hold different perspectives on the problem domain but express their needs in ways that often fail to conform to conventions of language use. The three bloggers in the airport security case study [4] illustrate this well. Even ignoring the divergence of semantics and pragmatics of their perspectives on the problem, a number of lexical and syntactic characteristics of the text pose real natural language

processing problems, such as idioms ('come on!'), implicit context ('we can deal with it.') and grammatical errors and typos ('We have to ban on ..', 'Channel No. 5').

The characteristics illustrated by the airport security blog illustrate why the automatic synthesis of user requirements is way beyond the current state-of-the-art. Useful support can be provided, however. A number of researchers have investigated the identification of domain concepts by analysis of the text using, for example, frequency profiling [5] and lexical affinities [6]. Such work can serve to help identify entities in the problem domain and their relationships, reveal key terms and populate glossaries. For example, in the airport security blog, the left hand pair of columns in Table 1 shows a ranked list of the ten words and their parts of speech that occur with a frequency that most exceed the frequency predicted by the 100 million word British national corpus (BNC). Note for instance that even though "oxidizer" appears only twice (once in singular and once in plural form) in the 603 word blog, twice is still significantly more frequently than predicted by its rate of occurrence in the BNC.

There are several interesting things to note here. The first is that "screen". "screening" and "screener" all share the same word stem so could have been collapsed into a single term. That hasn't been done because in the blog they represent sufficiently different concepts to make it worth distinguishing between them. Note that "screen" is a verb while the other two terms are nouns. Even "screener" and "screening", which are both nouns, are distinguished by the different semantic tags assigned by the tool we used to generate the data (Wmatrix [2]). "screening" is classified using the semantic tag A10 *Open/closed; Hiding/Hidden; Finding; Showing*. "screener" is classified as Z99 *Unmatched*. In other words, the semantic tagger failed to recognize "screener". Interestingly, there are six occurrences of "screening" in the text. Four are nouns and two are verbs. The verb form of "screening" is not as over-represented as the noun form so it does not appear in the top ten.

The two occurrences of "oxidizer" causing it to appear as the sixth most over-represented term illustrates why the application of statistical techniques to small volumes of text tends to yield results that should be interpreted with caution. The fact that a single blogger mentioned the term twice does not *per se* mean that it represents a significant concept within the bloggers' universe of discourse. That it *might* be significant can only be determined by a skilled analyst.

The third and fourth columns in Table 1 show the same as the first and second columns but this time, instead of restricting our analysis to the blog, we have included a small corpus of documents containing approximately 8000 words. This corpus was compiled from a mixture of press reports about airport security and advice on security published on travel websites. We cannot claim that it is truly representative of the domain. However, it is interesting to compare the first and third columns to help understand the focus of the blog within the general domain of airport security. If we had more confidence in the relevance and degree of consensus represented by the the corpus, we could use the results of the analysis as the starting point for the construction of a domain ontology that could be used for the reuse of knowledge across airport security applications. Given the degree of

uncertainty over the veracity of our corpus, the most we can claim in this instance is that it reveals some of the general context of the bloggers' conversation.

Table 1. The 10 Most Over-Represented Words in the blog and a domain corpus

Blog		**Blog + domain corpus**		**Blog + domain corpus**
Term	**PoS**	**Term**	**PoS**	**Verb**
screener	noun	airport	noun	access
security	noun	security	noun	screen
airport	noun	passenger	noun	check
administration	noun	new	adj	profile
government	noun	travel	noun	travel
oxidizer	noun	system	noun	identify
screening	noun	capta	noun	Travele *(sic)*
screen	verb	surveillance	noun	carry
tsa	noun	flight	noun	allow
ban	verb	luggage	noun	capture

The fifth column of Table 1 is also a ranked list extracted from the blog and the corpus. This time, however, we have filtered it on verbs to show only the action words. It is interesting to note that the verb "screen" reappears as a significantly over-represented action, as do "check", "profile", "identify" and other words related to the active application of security checks in airports.

In our earlier work [2], we elaborated on the use of statistical techniques for analyzing elicited requirements information. We analyzed the use of corpus-based frequency profiling, filtering on part-of-speech and on semantic class, and the use of lexical affinities. We concluded that frequency profiling yields the best performance of any individual technique. However, while generally performing relatively poorly when used in isolation, the other techniques tended to complement frequency profiling. Hence, for example, while frequency profiling will typically fail to identify some of the significant concepts present in a corpus of domain documents, at least some of the unidentified concepts will be statistically over-represented amongst the words that co-occur with the concepts that have been identified. Such co-occurrences show up as lexical affinities, so lexical affinity detection is a useful component of an analyst's toolkit which also supports frequency profiling. Similarly, other shallow NLP techniques are often able to provide a useful and complementary view of the information embodied in requirements text.

NLP techniques will never be capable of automating the derivation of requirements. Despite this, they have a role in assisting the human analyst's task of making sense of the myriad sources of information needed to inform the synthesis of user requirements. Whether NLP-assisted or not, information can be lost during the synthesis process, particularly when that information never existed in explicit form. The next section examines the role that shallow NLP techniques can play in recovering this lost information.

3 Upstream Trace Recovery

The process of user requirements synthesis is the first step in transitioning from the informal to the formal, although it is far from a simple activity and may involve (for example) goal modeling, scenario derivation, brainstorming and much else. Given the complexity of the process, it is good practice to record the synthesized requirements' motivation since maintenance of an explicit record helps inform trade-offs and allows backwards tracing to the stakeholders or information sources that motivated the requirements. Such upstream or *pre-requirements specification tracing* (pre-RST) [7] is, for a variety of reasons, commonly neglected.

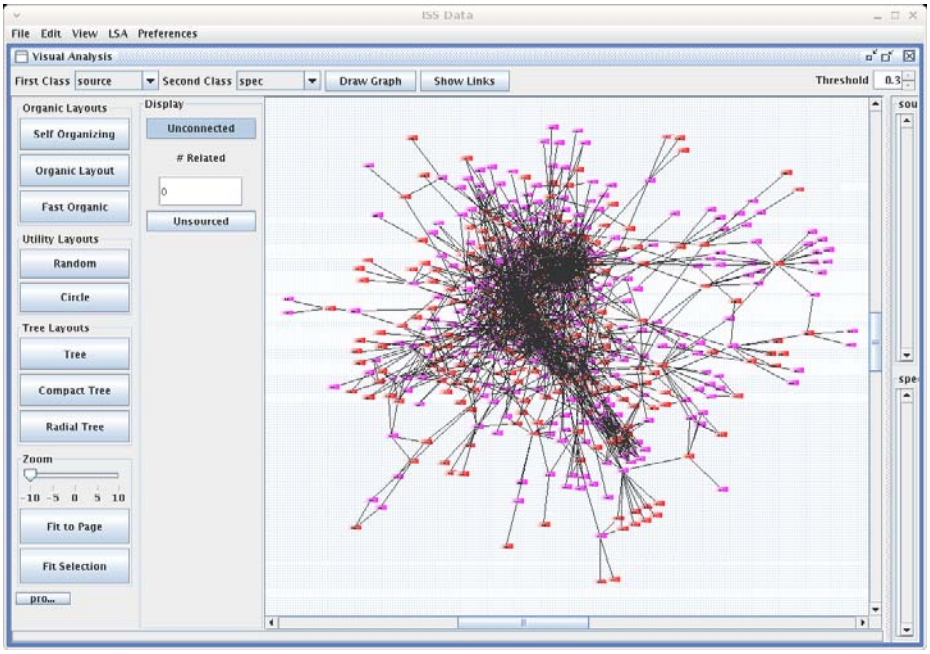


Figure 1. An Organic Layout Algorithm Used to Display Pre-RST *Derives* Relationships

Down-stream tracing (post-RST) is also commonly neglected, despite the ready availability of commercial requirements management (RM) tools that directly support down-stream tracing. This failure of basic RM practice has motivated several researchers to investigate automatic down-stream trace recovery. Techniques borrowed from information retrieval (IR) have been shown to be capable of inferring relationships between requirements at different levels of elaboration [8, 9, 10]. We have applied similar techniques to up-stream trace recovery using our *Prospect* tool [11].

The results of our evaluations indicate that the IR technique at the core of Prospect, latent semantic analysis (LSA) [12], is capable of inferring *derives* relationships between user requirements and the elicited knowledge that motivated them. In other words, a user requirement that participates in a semantic relationship with passages of elicited text is likely to have been motivated in some way by the information embodied by the elicited text. Figure 1. shows a cluster of several hundred requirements and passages of text from the information elicited form the stakeholders in a project. The scale is too small to see clearly here but requirements and "source" passages are represented as nodes with different colours. The arcs represent derives relationships and the tight clusters reveal where the multiplicity of relationships is high.

Most of the automatic trace recovery tools that have been developed use a measure of the lexical similarity between two requirements statements to infer semantic relatedness. The technique used by Prospect, LSA, also measures lexical similarities but is able to do more than count the number of co-occurring words. Prospect can infer a relationship between (say) a user requirement and passages of elicited information even when the terms used are somewhat dissimilar, provided that the terms that are used occur sufficiently commonly in similar contexts for LSA to infer synonymy or polysemy. Hence, LSA can recognize concepts than underlie lexical signifiers. For example, "airline" and "carrier" are often used as synonyms of the same underlying concept, and LSA offers a mechanism to recognize this without requiring the manual construction of a glossary.

In practice, Prospect achieves a level of recall and precision broadly consistent with the figures shown in Figure 2 which are derived from one of our case studies. Note that the "Threshold" value that calibrates the X axis represents the tool's adjustable sensitivity. The higher the threshold of similarity, the better the precision (i.e. fewer false positives) but the lower the recall (i.e. more valid relationships are missed). The technique will always produce false positives, but our experiments with user groups suggest that analysts can tolerate surprisingly high levels of imprecision in return for high recall.

We ran Prospect on the airline security case study [4]. The blog entries are not requirements, although they contain information that an analyst might use to inform the synthesis of requirements. Similarly, the corpus of documents is not representative of material an analyst would elicit from stakeholders, but might plausibly embody knowledge that an analyst could use to develop an understanding the problem domain. Relationships detected between blog entries and passages of corpus text do not represent the *derives* relationships that Prospect was designed to identify, but other forms of relationship might be expected to exist. Prospect detected significant semantic relationships between six of the thirteen blog entries and passages of text from the corpus. A significant number of passages from the corpus showed a relationship with the first blog entry:

"We have to ban on airplane passengers taking liquids on board in order to increase security following the recent foiled United Kingdom terrorist plot. We are also working on technologies to screen for chemicals in liquids, backscatter, you know?"

One of the interesting things about this entry, and the blog as a whole, is that there is no explicit rationale for why passengers should be prevented from carrying liquids on board an airplane. However, the rationale *is* provided by several of the corpus passages that Prospect linked with the blog entry, including, for example:

"Claims that terrorists were plotting to use liquid explosives suggest they understood the limitations of current bomb detection methods, experts say."

Note that the common occurrence of "liquid"/"liquids" in the blog and corpus passages suggests that the relationship was inferred from lexical similarity only. This supports the good performance reported by techniques based purely on lexical similarity, such as [8] but offers no insight into the advantages of using the more computationally-intensive LSA.

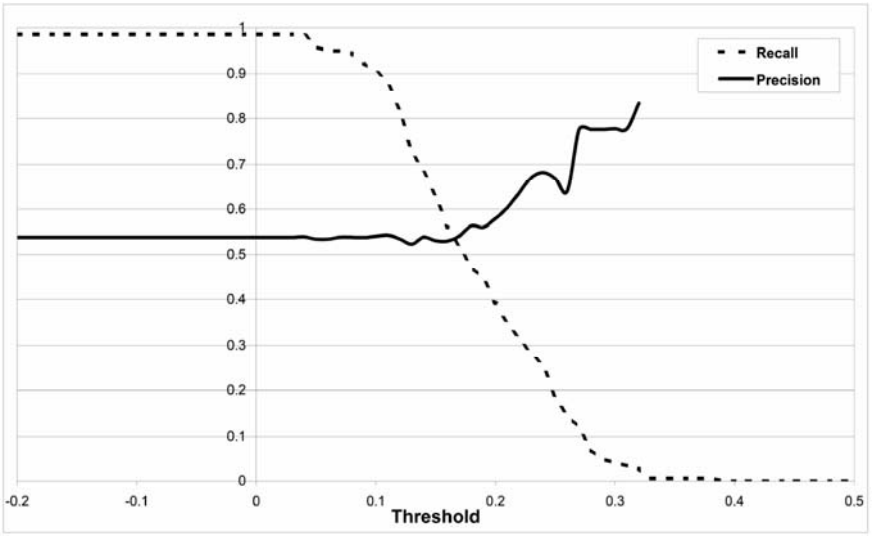


Figure 2. Recall and Precision Achieved by *Prospect*

LSA's tolerance of inconsistent vocabulary can be tested by our earlier observation that we would expect the synonymy of "airline" and "carrier" to be recognized. The corpus contains four passages of text that use the term "carrier" and many more that use "airline". Prospect identified relationships between two of the passages that used "carrier" with passages using "airline". That the recall was less than 100% reflects the fact that the weight that LSA attaches to two passages of text is proportional to the number of terms and concepts they share. A single shared concept such as that represented by "airline"/"carrier" is often insufficient in itself to show up as a strong degree of relatedness. This is not a failing of LSA; the fact that two passages of text mention either "airline" or "carrier" need not mean that they share deep semantic meaning. An example

of genuine semantic relatedness is illustrated by the following two passages of text that Prospect correctly inferred a relationship between:

"Travelers are urged to check with airlines in advance."

"Passengers are strongly advised to check the website of their carrier or airport before travelling."

Note that the semantic relatedness is revealed not only by the synonyms "airline" and "carrier", but also by the shared term "check" and another pair of synonyms: "traveler" and "passenger".

Our simple experiment confirmed our hypothesis that interesting semantic relationships would exist between blog entries and the corpus. In addition to offering an insight into the advantages of LSA over techniques based on purely lexical similarities, the experiment suggests that the utility of tools like Prospect might extend beyond trace recovery to provide more general assistance for analysts. However, there is one utility of tools such as Prospect that is revealed by performing trace recovery and which we explore in the next section.

4 Unprovenanced Requirements

An interesting phenomenon that is commonly revealed by applying Prospect to upstream trace recovery is that of *unprovenanced* requirements. If the elicited information exists in text form, Prospect if typically able to infer derives relationships between user requirements and passages of the elicited text. The strength of a relationship between a user requirement and passages of elicited text can vary according to the lexical similarities that exist between them, but with the tolerance of synonymy and polysemy that LSA affords. In the case studies conducted so far, a minority of requirements appear to have no relationship with the elicited text. The largest of our case studies was conducted on a live project and we were able to interview the analysts to validate the results. Their responses showed a strong correlation between requirements identified by Prospect as unprovenanced and those where the requirements had been "invented" by application of the analysts' domain knowledge.

Clearly, invention is part of the job of an analyst because they must use their knowledge and experience to creatively add value to the needs stated by the stakeholders. One common reason for the need for invention is that the information elicited from the stakeholders is incomplete. Incompleteness can be due to a number of reasons, but one is that the stakeholders hold information that they don't articulate either through deliberately withholding it or (we assume, more commonly) unconsciously withholding it. Knowledge that is never articulated, either because it is hard to articulate, or is so integral to the holder's model of the world that they don't feel the need to make it explicit is *tacit* [13, 14, 15].

A number of elicitation methods exist that help cope with tacit knowledge or concealed information [16]. EasyWinWin [17], for example, is designed to identify, refine and reach

consensus on the requirements for a system over a series of steps. These steps are carefully structured using prompts and the staged revelation of stakeholders' requirements and priorities to tease out concealed information. We hypothesize that techniques such as LSA could enhance tool support for such methods by, for example, tracing the evolution of stakeholders' requirements over stages in the elicitation process and helping highlight discontinuities that might be revealing of concealed information or tacit knowledge. Hence, in addition to helping detect the effect of tacit knowledge in sets of requirements, LSA may be useful in drawing tacit knowledge and concealed information out of stakeholders during requirements elicitation.

5 Conclusions

In [18], Kevin Ryan offered a critique of the application of natural language processing techniques to requirements engineering problems. Among Ryan's key observations was that it was both unfeasible and undesirable to automate the derivation of requirements from natural language text. Fourteen years later, Ryan's view still holds. Instead, work has focused on using NLP techniques as a tool to aid the human analyst. We argue that in the early stages of RE where the language is inevitably uncontrolled, shallow NLP techniques hold real promise as the basis for viable analysts' tools.

One of the reasons why the automation of the analyst's task is unfeasible and undesirable is that much of the information that the analyst needs in order to formulate appropriate requirements is likely to be unstated. We have described how latent semantic analysis, when applied to up-stream trace recovery can highlight disconnects between the formulated requirements and the information elicited from stakeholders. It appears that this disconnect is sometimes a symptom of missing or incomplete information, which in turn can be caused by stakeholders failing to articulate their knowledge. We believe that the ability to detect evidence of tacit knowledge is useful in itself and may form a component in a toolset for improving how tacit knowledge is handled within RE.

References

[1] http://fabrice.kordon.free.fr/Monterey2007/home.html
[2] Sawyer, P., Rayson, P., Cosh, K.: "Shallow Knowledge as an Aid to Deep Understanding in Early-Phase Requirements Engineering", IEEE Transactions on Software Engineering, 31 (11), November 2005.
[3] Yu, E. (1997) "Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering", Proc. Third IEEE International Symposium on Requirements Engineering (RE'97), Annapolis, MD. USA.
[4] http://fabrice.kordon.free.fr/Monterey2007/invitation_files/case-1.pdf

[5] Lecœuche, R. (2000) "Finding comparatively important concepts between texts", Proc. Fifteenth IEEE International Conference on Automated Software Engineering (ASE'00), Grenoble, France.
[6] Maarek, Y. and Berry, D. (1989) "The Use of Lexical Affinities in Requirements Extraction", Proc. fifth International Workshop on Software Specifications and Design, Pittsburg, Pa, USA
[7] Gotel, O., Finkelstein, A.: "An analysis of the requirements traceability problem", Proc. 1st International Conference on Requirements Engineering (ICRE'94), Colorado Springs, Co., USA, April, 1994.
[8] Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., Karlsson, J.: "A Feasibility Study of Automated Support for Similarity Analysis of Natural Language Requirements in Market-Driven Development", Requirements Engineering, 7 (1), 2002.
[9] Huffman-Hayes, J., Dekhtyar, A., Karthikeyan Sundaram, S.: "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods", IEEE Transactions on Software Engineering, 32 (1), January 2006.
[10] Cleland-Huang, J., Settimi, R., Romanova, E., Berenbach, B., Clark, S.: "Best Practices for Automated Traceability", IEEE Computer, 40 (6), June 2007.
[11] Stone, A., Sawyer, P.: "Identifying Tacit Knowledge-Based Requirements", IEE Proceedings Software, 153 (6), December 2006.
[12] Deerwester, S., Dumais, S., Furnas, G., Landauer, T., Harshman, R.: „Indexing by latents semantic analysis", J. Am. Soc. For Inf. Sci., 41 (6), 1990.
[13] Polanyi, M.: The Tacit Dimension, Peter Smith, Gloucester, Ma. USA, 1983.
[14] Nonaja, I, Takeuchi, H.: "A theory of organizational knowledge creation", Int. J. of Technology Management, 11 (7/8), 1996.
[15] Busch, P. and Richards, D.: "Acquisition of Articulable Tacit Knowledge", In Proc. Pacific Knowledge Acquisition Workshop (PKAW'04), in conjunction with The Eighth Pacific Rim International Conference on Artificial Intelligence, August 9-13, 2004, Auckland, New Zealand, 87-101.
[16] Collins, H.: "What is tacit knowledge", in Schtzki, T. Knorr, C. & von Savigny, E. (Eds) The practice turn in contemporary theory, Routledge, London and New York, 2001.
[17] Grünbacher, P., Briggs, R.: "Surfacing Tacit Knowledge in Requirements Negotiation: Experiences using EasyWinWin", Proc. 34th Hawaii International Conference on System Sciences, Hawaii, USA, 2001.
[18] Ryan, K.: "The Role of Natural Language in Requirements Engineering", Proc. First IEEE International Symposium on Requirements Engineering (RE'03), San Diego, Ca. USA, 1993.

A Case for ViewPoints and Documents

Michael Goedicke

Specification of Software Systems, ICB, University of Duisburg-Essen, Essen, Germany
Michael.Goedicke@icb.uni-due.de

Position Statement

In this contribution we view the airport security case study as multiple viewpoints in order to express the three stakeholders' views. Since there is an obvious "non convergence" in their views it is important to address the various sources of inconsistency between viewpoints. We advocate addressing not only "traditional" inconsistency to drive development forward but include other forms of imprecision like ambiguity and vagueness. In particular it is necessary to combine the viewpoint-oriented style with a document-oriented style of software development, serving management needs.

The Issues and Perspectives of the Case Study

The log representing the discussion between airport security related stakeholders is a typical exchange of ideas related to different layers of a given problem area. As usual the stakeholders perceive the problem area in different ways. Thus, the various utterances give rise to numerous problems to create a common understanding and – as we envisage the follow up of this discussion is not the creation of a set of agreed measures to improve air traffic security. If each stakeholder is required to write a summary of actions points as a result of the discussion quite diverging actions will arise. While the TSA representative addresses high level goals and deficiencies regarding political and security goals like "felt" security, the FAA representative is concerned with the financial performance of the processes and technologies employed and finally the ground personnel representative is concerned with the actual procedures and shortcomings of the human resources at hand.

Thus it is certainly clear that a view-oriented approach like the ViewPoint approach [1] is helpful in analyzing the situation sketched in the case study. The various stakeholders can be presented using ViewPoints and given an appropriate representation scheme – first order logic e.g. xlinkit [3] or temporal logic will certainly fulfill the task – inconsistencies can be made explicit. However, this is only the starting point.

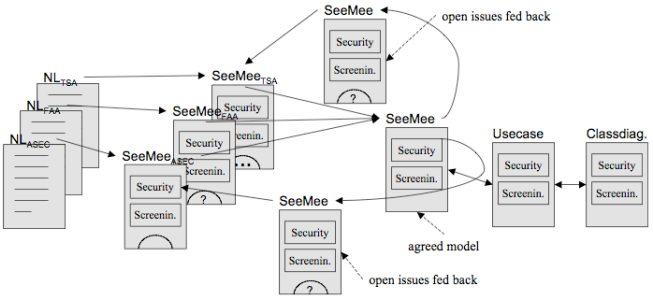
Michael Goedicke

As was put forward in [2] there has been the emphasis on living with inconsistencies and providing some repair actions if possible. The overall processes involving ViewPoints and related global properties (like convergence i.e. removal of essential inconsistencies) have to be elaborated and refined. Especially, if one regards requirements engineering as a set of continuous activities in parallel to creating other development artifacts, it is an important goal to integrate a document-oriented approach into the otherwise distributed views. Thus the challenge is to integrate document centric views – in many cases expressed today in some UML dialects or sublanguages – with more interaction centric views as represented by the blog of the three stakeholders of the case study.

While much of our work was concerned with the first type of integration and inconsistencies the latter is quite important and different approaches have to be used to achieve meaningful results from representing the stakeholders by different views explicitly. Earlier work [4] suggests that this is possible by modeling the involved layers (organization, informal relationships etc.) but additional results can be achieved if other forms of imprecise information are represented as well.

A contribution to this is if vagueness is represented explicitly as e.g. in the SeeMe approach [5][6]. The characteristic of the diagrammatic SeeMe approach is to introduce explicit graphical elements in order to indicate ambiguities, vagueness or other type of imprecision (ellipses, question marks, or arcs with unspecified source / target element in a diagram). Many of the statements recorded in the blog cannot be transformed directly into (UML-)specs but are more in the sense of recommendations, suggestions, assumptions or even (implicit) suspicions. In addition, work in the area of natural language processing (e.g. based on the work of Berry and Kamsties [7]) will be helpful to identify such sources of imprecision for the creation of SeeMee-models. Thus blogs as in the case study could be used in a (semi-)automatic way to derive SeeMee-Models which in turn can be fed into related ViewPoints using e.g. UML notations. The latter process can be possibly described by structural transformations.

The figure below sketches this process: ViewPoints using natural language (NL) are used to derive SeeMe-models (SeeMe) which are used to hammer out commonalities and differences based on the identified incomplete, vague and/or ambiguous pieces of specification. From the agreed model "normal" use cases and other types of precise specifications are derived and analyzed.



Finally an additional challenge is to define how – by enacting individual views – the overall process can be managed to generate progress. Obviously, this is not automatic since it needs the cooperation of the involved stakeholders to achieve progress.

References

[1] A. Finkelstein, A. Kramer, J. Nuseibeh, B. Finkelstein, L. Goedicke, M. *Viewpoints: A Framework for Integrating Multiple Perspectives in System Development* in: Intl. Journal of Software Engineering and Knowledge Engineering 2(1): 31-57, 1992

[2] Nuseibeh, B. Kramer, J. Finkelstein, A. *Viewpoints: meaningful relationships are difficult!*, Proc. ICSE'03, Portland, Oregon, USA, May 2003, IEEE CS Press.

[3] Nentwich,C. Capra, L. Emmerich, W. Finkelstein, A. *xlinkit: a Consistency Checking and Smart Link Generation Service*, ACM Trans. On Internet Technology, 2(2) 151 – 185, May 2002

[4] Piwetz, C. *Requirements Definitions for Groupware Systems – A View-Oriented Approach*, Dissertation, Univ. Duisburg-Essen, 2001

[5] Herrmann, Th. Loser, K.-U. Vagueness in models of socio-technical systems. In: Behaviour and Information Technology. Vol. 18, no. 5, pp 313-323, 1999

[6] Herrmann, T. *SeeMee in a Nutshell*, Technical Report Univ. Bochum, 2006 https://web-imtm.iaw.ruhr-uni-bochum.de/pub/bscw.cgi/0/208299/30621/30621.pdf

[7] Berry, D.M. Kamsties, E. *Ambiguity in Requirements Specification* in Perspectives on Software Requirements, Leite, J. Dorn, J. (EDs), Springer, 2003

Getting the Details Right

Lori A. Clarke
Department of Computer Science
University of Massachusetts
Amherst MA 01003

Abstract

Requirements specifications are usually long, natural-language documents, which in addition to describing the high-level functionality of the subject system, provide a relatively long list of "shall" or "should" conditions, which we refer to as property specifications. These informal property specifications are often the basis for formally specified property specification that can be subsequently used in the verification of the system design or implementation or in test planning and test case generation. This talk will describe our experiences going from high-level informal descriptions in natural language to mathematically precise specifications using PROPEL, a system that helps in the elicitation and representation of the many details that must be considered. Examples will be drawn from the UMASS Medical Safety Project, where medical professionals are working with computer scientists to define and improve life-critical medical processes.

Model-Driven Prototyping Based Requirements Elicitation

Jicheng Fu, Farokh B. Bastani, and I-Ling Yen

Department of Computer Science
The University of Texas at Dallas
P.O. Box 830688, EC 31
Richardson, TX 75083-0688
{jicheng.fu@student.utdallas.edu, bastani@utdallas.edu, ilyen@utdallas.edu}

Abstract

This paper presents a requirements elicitation approach that is based on model-driven prototyping. The model-driven approach fits naturally in evolutionary prototyping because modeling and design are not treated merely as documents but as key parts of the development process. This paper proposes a component-based program synthesis technique to automate business/logic code synthesis. It has a sound theoretical basis and is able to generate reliable software. Combined with transformation, our approach not only facilitates rapid prototyping, but also achieves high reliability.

Keywords: Requirements Elicitation, Prototyping, Component-Based Software Development, Code Patterns, Model-Driven Development

1 INTRODUCTION

The primary measure of success in a software system is the degree to which it meets the purpose for which it is intended [1]. Therefore, requirements engineering activities are vital in ensuring successful projects.

Prototyping is a popular requirements elicitation technique and is especially useful when there is a great deal of uncertainty or when early feedback from stakeholders is needed [2]. Prototypes can concretely present the system operation and facilitate design decisions. Therefore, the primary advantages of prototyping are reduced time and cost, and more user involvement. However, prototyping has some disadvantages, such as requirements that are not traceable and the potential for patchwork programs [3].

Recently, use cases have become popular for capturing functional requirements. Use cases document initial requirements and provide scenarios illustrating interactions with end users or other systems to achieve specific business goals. However, use cases are not good for capturing non-functional requirements [3].

The use of model-driven approaches in requirements engineering is especially amenable to model-driven development (MDD) process and agile software development. UML 2.0, developed to support MDD, has changed the view that UML diagrams only serve as temporary documents and will be put aside at later points during development. Combined with OCL (Object Constraint Language), UML is able to specify models in a formal way. OCL is a declarative and precise specification language, which has no side-effects and does not change the state of the system [4]. It enables errors to be found early in the life-cycle, when fixing a fault is relatively cheap.

Traceability is another helpful asset of model-driven development. It makes the development process amenable to requirement changes. It is always easier to indicate what part of a Platform Independent Model (PIM) is affected by the changed requirements than to determine code segments that must be modified. When parts of the code can be traced back to elements in the PIM, it would be much easier to make an impact analysis of the requested changes [5].

The proposed approach is based on an iterative and evolutionary model-driven development process. It combines the advantages of prototyping and use cases in requirements engineering. Transformations that map models to the next level are typically used in Model-Driven Architecture (MDA) [9] and Model-Driven Software Development (MDSD) [6]. However, there are some doubts about the practicality of generating complete systems solely via

transformations. For example, MDSD only generates infrastructure codes. The program synthesis technique involved in the proposed approach combines transformational and component-based synthesis techniques to automatically generate business/logic codes. This can facilitate evolutionary model-driven prototyping and greatly accelerate the development process.

The rest of this paper is organized as follows: Section 2 introduces a component-based program synthesis method. Section 3 illustrates how model-driven based development approach is used to elicit the requirements. Section 4 concludes the paper.

2 COMPONENT-BASED PROGRAM SYNTHESIS

To support Component-Based Software Development (CBSD), it is necessary to reduce the learning curve and automate the composition process. A method along this direction is pattern-based code synthesis in which code patterns [10][11] are used to capture the typical usages of the components as well as their interactions. Six composition operators, including one instantiation operator (Map), three functional operators (Concatenate, Invert, and Splice), and two non-functional operators (distribution and exception handling), have been formally defined for glue code synthesis.

Based on the original Code Pattern Integration System (CPIS) [11] that enables users to compose code patterns and achieve semi-automated synthesis of the glue code, an approach has been proposed [12][13] to automate the code pattern composition process.

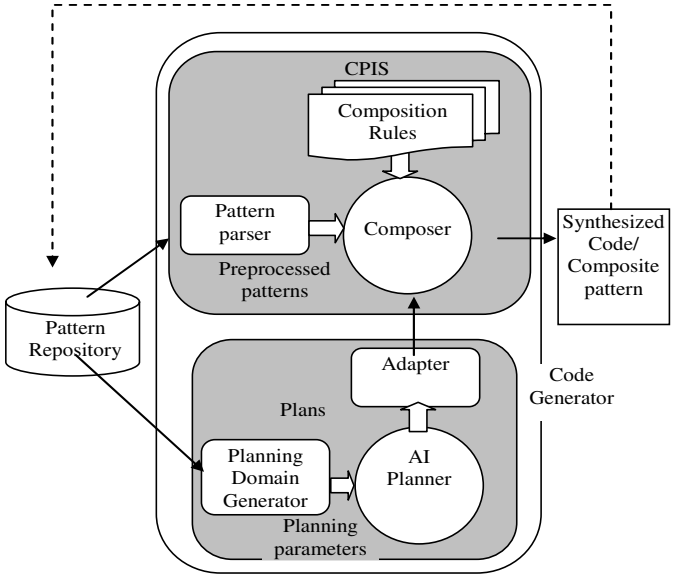


Fig. 1 Architecture of code pattern based automated code synthesis system

As shown in Fig. 1, the Code generator consists of two components, namely, Code Pattern Integration System (CPIS) and Planning System. The planning system is the core in which the AI planner is able to generate parameterized procedure-like generic reusable plans called *procedural plans*. The planning domain generator maps code patterns from the pattern repository into operators of the planning domain. The procedural plans are then translated into preprocessed patterns that are fed to the composer to synthesize composite code patterns.

3 REQUIREMENTS ELICITATION via PROTOTYPING

Fig. 2 illustrates the development cycle of the proposed approach. In model-driven development, stakeholders and developers are actively involved in the whole development process. Use cases are the first tangible things that stakeholders interact with and work closely with developers to achieve agreement. The requirements specification process can be automated using tools [7][8] that capture essential and relevant software requirements from natural language descriptions. These tools employ natural language processing (NLP) techniques and can produce UML elements. This can help automate Object-Oriented Analysis (OOA) though the tools are not mature and only aid the requirements acquisition and analysis process. Human involvement is mandatory especially when contradictions exist in the requirement specification, e.g., the client might initially propose detecting all carry-on liquids on commercial flights, but may subsequently suggest treating it differently based on the volume as the elicitation process proceeds [15].

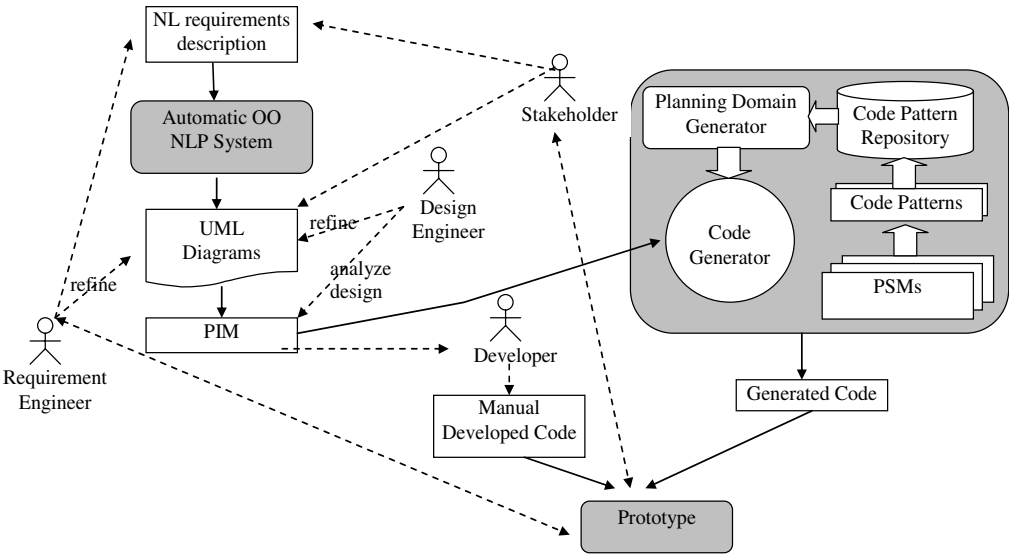


Fig. 2 Model-Driven Prototyping

3.1 Context Specification

We apply the context assessment during the requirements acquisition phase. This includes the precise specifications of functional and non-functional requirements, of platform constraints, and of operational environment and threats.

Functional requirements specify processes that the system has to perform or information it needs to contain [14]. For the air traveling case study [15], the functional requirements specify that automated screening devices should be able to detect oxidizers and, in the long run, the system should be able to predict possible means of attacks and come up with solutions to prevent them from happening, etc. However, it is not clear at this point whether the case study is targeting a software system, a hardware system, or just coming up with some manual regulations. If this is for a software system, the proposed method is well suited to capture the requirements because prototypes are especially useful when there is a great deal of uncertainty or when early feedback from stakeholders is needed [2].

Non-functional requirements address behavioral properties that the system must have [14]. For example, some false alarms of the screening devices are tolerable. However, excessive false alarms imply poor QoS of these devices. Maintainability is a major concern for the air traveling example as the screening rules may change. The system should be designed to be easily upgradeable to deal with these changes. Moreover, cost issue is also an important aspect. Just as the air traveling example shows, “an economic threat is also a threat”. In practice, it is important to demonstrate adequate ROI (Return on Investment) for the money that has been spent to build, deploy, and operate the system.

Specification of platform constraints introduces additional non-functional constraints on the system. With regard to the air traveling case study [15], it is not clear that the goal of the Blog discussion is for a software system, hardware system, or manual operations. Hence, the specification of platform capabilities (actuators and sensors) and constraints (memory, power, weight, size constraints, etc.) cannot be obtained.

Specification of the operational environment and potential threats capture both the impact of the environment on the system as well as the impact of the system on the environment. For the air traveling example, it includes:

- Operators: the individuals who are responsible for deploying and maintaining the daily operation of the system. They may also decide when to conduct "surprise tests" to keep the screeners alert and whether to send some screeners for training if they fail the tests.
- Users: the persons who use the system, e.g., screeners.
- Adverse physical impacts of the environment on the system. This is typically due to environmental events that affect the physical platform. For example, the damage of a screening device can affect the availability of the system. Typically, redundancy is the key solution to such requirements.

In general, the specification of the operational environment and threats results in a detailed specification of the non-functional requirements, e.g., availability, safety, security, usability, etc.

3.2 Iterative Process

The proposed model-driven development process is different from MDA in the sense that there are no transformations from PIMs to PSMs (Platform Specific Models). This shares some similarities with the MDSD approach [6], in which no PSMs are necessary also. Practical project experience has proved that this simplification is usually more useful than the additional degree of freedom gained with PSMs. As a result, there is no need to control, manipulate, and enrich the various intermediate transformation results with specific information. This not only allows for more efficient development, but also avoids potential consistency problems: a manual change of an intermediate model might result in an inconsistency with higher abstraction levels that is not automatically correctable [6].

In model-driven development, the focus shifts to PIMs, which are higher level abstractions than programs. When requirements change, it is often easier to indicate what part of the PIM is affected by the changed requirements than to identify which part of the code must be adapted. The PIMs are easily understood by developers and, hence, code quality can be improved as the development cycle iterates.

The difference between the proposed approach and MDSD is that besides transformation, it uses the component-based method to synthesize business/logic codes by automating the code pattern operations. Code pattern is a higher level abstraction of platform specific components and is used to capture the typical usages of these components as well as their interactions. The program generator is based on an enhanced AI planner which is able to generate parameterized procedure-like generic reusable plans (procedural plans).

The component-based program synthesis method can be seamlessly integrated into the proposed model-driven approach. In the PIM part, OCL is used to express constraints and, hence, make the components complete, consistent, and unambiguous. Among the constraints, the pre-condition of a method is used to specify the conditions where the method can be applied, while the post-condition of a method indicates the expected outcomes. As discussed in Section 2, code patterns have a sound formal basis using OCL and can be converted into planning operators so that an AI planner can work on the code patterns and derive a procedural plan. This fits naturally in the definition of the planning problem, $P = (O, s_0, g)$, where O is a collection of operators, s_0 is the initial state, and g is the goal state. Under the context of the proposed MDD approach, O is the set of planning operators that are converted from the code pattern repository; s_0 and g are presented by constraints specified in PIMs. The AI planner in the program generator takes the planning problem as input and generates a procedural plan which is then transformed into low-level code. Developers have the flexibility to specify constraints at the method level or embed constraints at lower levels, e.g., method bodies. They then just focus on incomplete parts where the planner cannot find a suitable solution. This can alleviate the developers' burden and increase the development speed and the reliability of the system.

After the system is complete, stakeholders can visually operate it and formulate new requirements to cope with any problems. These will be fed back to requirement engineers and the development cycle is repeated.

4 CONCLUSIONS

We have proposed a model-driven prototyping approach for requirements engineering. It inherits the advantages of prototyping elicitations without the disadvantages, such as untraceable requirements and patchwork programs [3], by applying model-driven development principles and advanced program synthesis techniques. The proposed approach is an evolutionary process that iteratively refines the requirements, design, and implementation and yields high quality systems with the help of the proposed component-based program synthesis technique.

REFERENCE

[1] Nuseibeh B, and Easterbrook S: "Requirements Engineering: A Roadmap", The Future of Software Engineering, Special Issue 22nd International Conference on Software Engineering, ACM-IEEE, 2000, pp: 35 – 46.
[2] A. Davis: "Operational Prototyping: A New Development Approach", *Software*, September/October 1992, Vol. 9, No. 5, pp: 70 – 78.
[3] Wikipedia: "Software Prototyping", *http://en.wikipedia.org/*.
[4] J. Warmer, A. Kleppe: "The object constraint language: getting your models ready for MDA", *Addison-Wesley*, 2003.
[5] A. Kleppe, J. Warmer, and W. Bast: "MDA explained: the model driven architecture: practice and promise", *Addison-Wesley*, 2003.
[6] T. Stahl, M. Völter, J. Bettin, A. Haase and S. Helsen: "Model-driven software development: technology, engineering, management", *John Wiley*, 2006.
[7] H. M. Harmain and R. Gaizauskas: "CM-Builder: A Natural language-based CASE Tool", *Journal of Automated Software Engineering*, 10, 2003, pp: 157 – 181.
[8] S.L.V. Overmyer and O. Rambow: "Conceptual Modeling through Linguistics Analysis Using LIDA", 23*rd international conference on Software engineering*, July 2001.
[9] Object Management Group: "MDA Guide: Version 1.0.1", *OMG document omg*/03-06-01, 2005.
[10] J. Liu, F. B. Bastani, and I. Yen: "Code Pattern: An Approach for Component-Based Code Synthesis", *Proceeding of the 7th World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL, July 2003, pp: 330 – 336.
[11] J. Liu, F. B. Bastani, and I. Yen: "A Formal Foundation of the Operations on Code Patterns", *The International Conference on Software Engineering and Knowledge Engineering*, Taipei, Taiwan, Republic of China, July 2005.
[12] J. Fu, F.B. Bastani, and I. Yen: "Automated AI Planning and Code Pattern Based Code Synthesis", *ICTAI* 2006, pp: 540 – 546.
[13] J. Fu, F.B. Bastani, and I. Yen: "Iterative Planning in the Context of Automated Code Synthesis", *COMPSAC* 2007, pp: 251 – 259.
[14] A. Dennis, B. H. Wixom, D. and Tegarden, Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach, *Wiley*, 2 edition (August 10, 2004)
[15] http://fabrice.kordon.free.fr/Monterey2007/invitation_files/case-1.pdf

Improving the Quality of Requirements Specifications via Automatically Created Object-Oriented Models

Daniel Popescu[1] Spencer Rugaber[2] Nenad Medvidovic[1] Daniel M. Berry[3]

[1]Comp. Sci. Department
Univ. of Southern California
Los Angeles, CA, USA

[2]College of Computing
Georgia Institute of Tech.
Atlanta, GA, USA

[3]Cheriton School of Comp. Sci.
Univ. of Waterloo
Waterloo, ON, Canada

Abstract. In industry, reviews and inspections are the primary methods to identify ambiguities, inconsistencies, and under specifications in natural language (NL) software requirements specifications (SRSs). However, humans have difficulties identifying ambiguities and tend to overlook inconsistencies in a large NL SRS. This paper presents a three-step, semi-automatic method, supported by a prototype tool, for identifying inconsistencies and ambiguities in NL SRSs. The method combines the strengths of automation and human reasoning to overcome difficulties with reviews and inspections. First, the tool parses a NL SRS according to a constraining grammar. Second, from relationships exposed in the parse, the tool creates the classes, methods, variables, and associations of an object-oriented analysis model of the specified system. Third, the model is diagrammed so that a human reviewer can use the model to detect ambiguities and inconsistencies. Since a human finds the problems, the tool has to have neither perfect recall nor perfect precision. The effectiveness of the approach is demonstrated by applying it and the tool to a widely published example NL SRS. A separate study evaluates the tool's domain-specific term detection.

1 Introduction

The typical industrial software specifier writes software requirements specifications (SRSs) in a natural language (NL). Even if a final SRS is written in a formal language, its first draft is usually written in a NL. A NL SRS enhances the communication between all the stakeholders. However, on the downside, often a NL SRS is imprecise and ambiguous [6].

Many an organization follows a three-step review process to assess the quality of a NL SRS and to identify ambiguities and other defects in the NL SRS [1]. First, assigned reviewers try to find defects in the document. Second, in a meeting of the reviewers, all found defects are collected and rated according to their severities. Third, the reviewed NL SRS and the collected defects are sent back to the authors for corrections.

In this process, the quality of a review of a document is dependent mainly upon how effectively each human reviewer is able to find ambiguities and other defects in the document. However, a human reviewer and even a group of them have difficulties identifying all ambiguities, even when using aids such as checklists. A human reviewer might overlook some defects while reading a SRS, because he might assume that the first in-

terpretation of the document that came to his mind is the intended interpretation, unaware of other possible understandings. In other words, he *unconsciously disambiguates* an ambiguous document [11].

When a NL SRS is large, some ambiguities might remain undetected, because ambiguities caused by interaction of distant parts of the NL SRS are difficult to detect. In any one review sessions, only an excerpt of the NL SRS can be reviewed. Any ambiguity in the reviewed excerpt that is caused by interaction with a part of the NL SRS outside the excerpt may not be detectable. Moreover, when a NL SRS is large, lack of time may prevent some parts of the NL SRS from ever being reviewed. Having a faster method for conducting reviews would permit larger chunks of the whole NL SRS to be reviewed at once. It would permit also faster reviews so that more reviews and, thus, greater coverage of the NL SRS would be possible in any duration.

A controlled, grammar-constrained NL [9][10] helps to reduce ambiguities by constraining what can be said in the NL. One possible grammar rule eliminates passive voice and, therefore, ensures that the doer of an action is always known. However, a controlled NL can address only syntactic ambiguity. Semantic ambiguity is beyond its control.

Because a semantic model omits unnecessary details, it helps to reduce complexity, and it helps the user to visualize important aspects. Therefore, with the help of a semantic model, a human reviewer can validate larger system descriptions than otherwise. Moreover, the human reviewer can focus on conceptual correctness and does not need to worry about the consistent use of concepts and correct grammar usage. Therefore, if a semantic model can be created of the system specified by a NL SRS, a thorough review of the NL SRS becomes easier.

Of course, constructing a model bears the risks of introducing new defects and of the model's not representing the original NL SRS. Researchers have tried to mitigate these risks by using automatic approaches and NL processing (NLP) [28][24][12] techniques. A software tool can scan, search, browse, and tag huge text documents much faster than a human analyst can. Furthermore, a tool works rigorously, decreasing the risk of overlooked defects and unconscious disambiguation. Each of some of the approaches tries to build a model from a NL source and may even try to reason about the content, possibly with assistance from the human user.

In any automatic approach, the recall and precision of its NL parsing and of its domain-specific term (DST) identification bound the quality of the approach. The typical domain has its own terms, abbreviations, and vocabulary. Therefore, a tool must detect these to create a correct model. Each of some approaches has difficulty to create a correct model because it relies on a semantic network that is based on a predefined domain dictionary. For many domains a domain dictionary does not exist. Each of other approaches requires the human specifiers to build a domain dictionary while writing the NL SRS. Clearly, not every requirements engineering (RE) process is so mature that in it, a domain dictionary is built. Even in a mature process, some domain terms might be forgotten, because the analysts assume that these terms are widely understood and recognized.

We have created an approach for helping a specification writer or reviewer identify ambiguities in a NL SRS, in which the approach tries to address all of the above mentioned problems. Hereinafter, the new approach is called "our approach" to distinguish it from other approaches. For our approach, we have built a prototype *dowsing*[1] tool, called "Dowser". Dowser is based on one controlled NL. It can create from any NL SRS an object-oriented (OO) diagram, which can then be assessed by human reviewers.

In the first step, Dowser parses a NL SRS and extracts the classes, methods, and associations of a textual class model from the NL SRS. In the second step, Dowser diagrams the constructed textual class model. In the third step, a human reviewer can check the generated diagram for signs of defects in the NL SRS. Dowser and our approach are based on NL processing (NLP) and not on NL understanding. Dowser cannot judge whether or not the produced model describes a good set of requirements or classes. This judgement requires understanding. Therefore, in our approach, a human is the final arbiter of ambiguity. Because the human is in the loop, Dowser has to have neither perfect recall nor perfect precision.

To evaluate the effectiveness of our approach, we have implemented a prototype of Dowser and have tested it on a widely used example SRS, describing an elevator system [15]. The case study demonstrates the effectiveness of the approach. Since identifying domain terminology is required for any successful NLP-based approach, we conducted separate studies to evaluate Dowser's DST detection. The studies show that our approach is capable of achieving high recall and precision when detecting DSTs in a UNIX manual page.

Section 2 discusses related work. Section 3 describes out approach and all of its components. Section 4 describes validating case studies, and Section 5 concludes the paper by discussing the results and future work.

2 Related Work

Related work can be divided into four parts: (1) NLP on SRSs, (2) controlled languages, (3) automatic object-oriented (OO) analysis model (OOAM) extraction, and (4) domain-specific term (DST) extraction.

NLP on SRSs. Kof describes a case study of the application of NLP to extract and classify terms and then to build a domain ontology [18]. This work is the most similar to our approach. The built domain ontology consists of nouns and verbs, which constitute the domain's concepts. In the end, the domain ontology helps to detect weaknesses in the requirements specification. Gervasi and Nuseibeh describe their experiences using lightweight formal methods for the partial validation of NL SRSs [12]. They check properties of models obtained by shallow parsing of natural language requirements. Furthermore, they demonstrate scalability of their approach with a NASA SRS.

Controlled languages. Fuchs and Schwitter developed Attempto Controlled English (ACE) [9], a sublanguage of English whose utterances can be unambiguously translated

1. A dowser is a tool that makes use of domain knowledge in understanding software artifacts [7].

into first-order logic. Over the years, ACE has evolved into a mature controlled language, which is used mainly for reasoning about SRSs [10]. Juristo *et al.* developed other controlled languages, SUL and DUL [17]. For these languages, they defined a correspondence between linguistic patterns and conceptual patterns. After a SRS has been written in SUL and DUL, an OOAM can be created using the correspondence.

OOAM Extraction. Several tools exist that automatically transform a SRS into an OOAM. Mich's NL-OOPS [23] tool first transforms a parsed SRS into a semantic network. Afterwards, the tool derives an OOAM from the semantic network. Delisle, Barker, and Biskri implemented a tool that uses only syntactic extraction rules [4]. Harmain and Gaizauskas implemented another syntax-based tool, and they introduce a method to evaluate the performance of any such tool [15]. As in our approach, Nanduri and Rugaber [27] have used the same parser and had the same initial idea of using syntactic knowledge to transform a NL SRS into an OOAM. The objective of their approach was to validate a manually constructed OOAM. Our approach's main objective is to identify ambiguity, inconsistency, and underspecification in a NL SRS. The more restricted objectives of our approach enables a more detailed discussion of the problem space and contributes (1) a constraining grammar, (2) analysis interpretation guidelines, (3) additional transformation rules, and (4) DST extraction.

DST extraction. Mollá *et al*. developed a method for answering questions in any specified technical domain. This work recognizes the importance of dealing with specified technical terminologies in NLP tools that are applied to SRSs [26].

3 Our Approach

The goal of our approach is to reduce the number of ambiguities, inconsistencies, and underspecifications in a NL SRS through automation. Assuming that automation will not be perfect, we let a human make the final decision about a potential ambiguity, inconsistency, or underspecification.

While reading through a NL SRS, an engineer usually builds a mental model of the described system and reasons about the correct relations of the SRS's concepts. If an engineer could only analyze the correctness of a model, instead of having also to create it, write it down, and then analyze it, he could use his skills and time more effectively.

Considering that a reviewer could more effectively inspect a model than the complete NL SRS, we developed an automatic approach based on the following observations:

- Each of most software companies uses a NL SRS [22] to describe a software system regardless of its domain.
- An OOAM is able to show the most important concepts and relations among the concepts of a system to build.
- Many an OO design method suggests building an OOAM of a sentence by identifying the parts of the sentence and creating a class from the subject, attributes from the adjectives, and methods and associations from the verb [2][29].

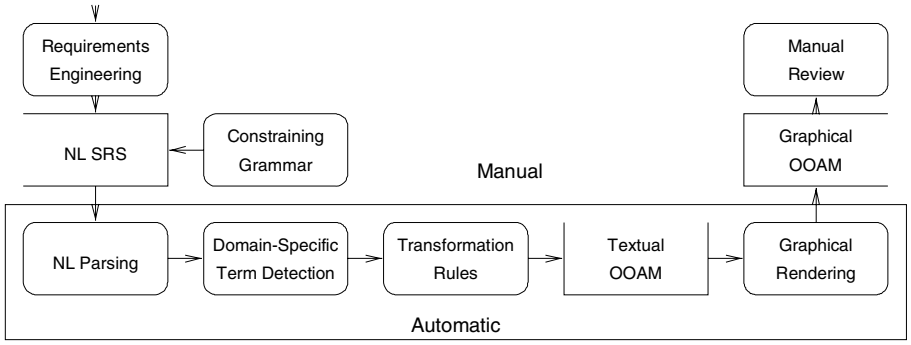
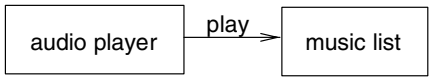


Figure 1. Flow of the Approach

Consider the example transformation of the sentence[1] The audio player shall play the music list. into an OOAM; audio player is the subject of the sentence, play is the verb, and music list is the direct object. This sentence could therefore be modeled:



In this example, the adjectives audio and music are not broken out, because each is part of a DST.

Using this syntax-based method, the typical functional requirement sentence of a NL SRS can be transformed into an OOAM. Since this heuristic suggests using mostly syntactic information, the transformation can be automated. A NL parser can create parse trees of any NL text [30]. The OO design literature gives many rules and heuristics to transform many a syntactic construct into a textual OOAM. Off-the-shelf software exists to diagram textual OOAMs [31].

Since NL SRSs are written for a wide range of domains such as medical, technical or judicial domains, a successful approach must be robust in identifying DSTs. Our approach addresses this need by using syntactic information and a robust parser with guessing capability.

The overall quality of any NL SRS can be improved by enforcing the use of a constraining grammar. A constraining grammar reduces the possibilities of ambiguities by constraining the allowed language constructs. At the same time, it increases the quality of parsing, reduces the number of parses, and results in more precise OOAMs.

---

1. A sans serif typeface is used for example text, except in a parse tree. Beware of punctuation, also typeset in the sans serif typeface, at the end of any example. It should not be considered as punctuation in the containing sentence, which is typeset in a serifed typeface. A typewriter typeface is used for example text in any parse tree, in which monospacing is essential for the correct display of the tree.

Therefore, by using and extending existing technology, we can create a tool that automatically transforms a NL SRS into an OOAM that helps a human being to identify ambiguities, inconsistencies, and under specifications in the NL SRS.

Figure 1 shows the flow of our approach. First, Dowser parses a NL SRS according to a constraining grammar. Second, from relationships exposed in the parse, Dowser creates the classes, methods, variables, and associations of an OOAM of the specified system. Third, the OOAM is diagrammed so that a human reviewer can use the model to detect ambiguities, inconsistencies, and underspecifications.

3.1 Constraining Grammar

Any NL allows expressing the same concept in different ways using different syntactic structures. For example, a sentence in active voice can be translated into passive voice without changing the semantics or pragmatics. However, passive voice can encourage ambiguities. For example, the sentence The illumination of the button is activated. leaves room for different interpretations, because it is not clear who holds the responsibility for activating the illumination. Alternatively, the sentence could be describing a state. As a consequence, a constraining grammar can be introduced to decrease the possibility of ambiguity. A constraining grammar enables formal reasoning without the disadvantages of a fully formal language [10]. A constraining grammar has the other advantage that it is more amenable to parsing, and extraction rules based on it can be created more easily.

Our approach uses a constraining grammar that is derived from Juristo *et al.*'s grammar [17]. They have developed two context-free grammars and an unambiguous mapping from these grammars to OOAMs. This mapping is explicitly defined and allows better model creation than with commonly used heuristics that are justified only intuitively. Moreover, the explicit definition enables automation.

Using a constraining grammar influences the style of a NL SRS, because a constraint grammar enforces simple sentences. The typical sentence has a basic structure consisting of subject, verb and object. Furthermore, only simple subclause constructions are allowed, such as conditional clauses, using if, when, velc[1]. Therefore, a NL SRS will contain many short sentences if it is written according to the developed controlled grammar. Shorter, simpler sentences tend to be less ambiguous, because at the very least, they avoid some coordination and scope ambiguities

3.2 Natural Language Parsing

Since an OOAM is created automatically from syntactic information, we needed a parser to extract this information from the NL SRS. The parser we used was developed by Sleator and Temperley (S&T) at Carnegie-Mellon University [30]. Sutcliffe and McElligott showed that the S&T parser is robust and accurate for parsing software manuals [33]. Since software manuals are similar to SRSs [5], the S&T parser looked promising for our approach. Additionally, the S&T parser was chosen because it is able to guess

1. "velc." means "or others" and is to "vel cetera" as "etc." is to "et cetera".

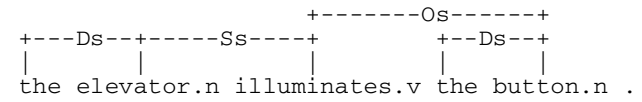
the grammatical role of unknown words. This capability is used for the DST detection, which is described in Section 3.4

The parser is based on the theory of link grammars, which define easy-to-understand rule-based grammar systems. A link grammar consists of a set of words, i.e., the terminal symbols of the grammar, each of which has one or more linking requirements. A sequence of words is a sentence of the language defined by the grammar if there exists a way to assign to the words links that satisfy the following three conditions:

1. Planarity: the links do not cross;
2. Connectivity: the links suffice to connect all the words of the sequence together; and
3. Satisfaction: the links satisfy the linking requirements of each word in the sequence.

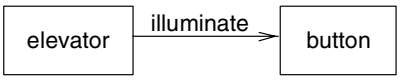
The link grammar parser produces the links for every such sentence. After parsing, the links can be accessed through the API of the link grammar parser.

Each established link in a sentence has a link type, which defines the grammatical usage of the word at the source of the link. The sentence The elevator illuminates the button. shows three different link types:



A `D` link connects a determiner to a noun, an `S` link connects a subject noun to its finite verb, and an `O` link connects a transitive verb to its object. A small `s` after the type of a link indicates that the target of the link is a singular noun.

From this example sentence, the first extraction rule can be derived. If a sentence contains an `S` link and an `O` link, then create a class from the subject noun and one from the object noun. Afterwards, create a directed association from the subject class to the object class, which is named by the verb:



A directed association was chosen over a simple class method, because a directed association shows also who invokes the action. If the action were modeled as a class method, the information about who causes the action would have been lost. Using this rule, Dowser would extract the classes elevator and button and a directed association illuminate from the elevator class to the button class.

To avoid having two different classes created for elevator and elevators, our approach incorporates the lexical reference system *WordNet* [25] to find the stems of nouns and verbs. Therefore, the name of each created class is the stem of a noun.

One might think that the use of a constrained language would reduce the number of parses and might even ensure that there is only one per sentence. However, the language

constraints only encourage and do not guarantee uniguity[1]. In general, the link parser returns multiple parses for any input sentence. For example, When an elevator has not to service any requests, the elevator remains at its final destination and the doors of the elevator are closed. returns 16 parses. However, as the link grammar homepage [34] says, "If there is more than one satisfactory linkage, the parser orders them according to certain simple heuristics." In our experience, these heuristics selected as first our preferred parse. Of courser, these simple heuristics cannot be expected to always select the parse that the writer intended, because syntactic ambiguity is a hard problem. Even humans have to rely on semantic understanding and context to resolve difficult cases of syntactic ambiguity, e.g., the classic The boy saw the man with the telescope..

Therefore, Dowser was consciously designed to help the user see semantic ambiguities and ignores dealing with syntactic ambiguities even though it sees multiple parse trees, because

1. there are several tools that deal with syntactic ambiguities (e.g. [35][19][3]) but
2. there are very few, if any, that deal with or even just help deal with true semantic ambiguities.

3.3 Transformation Rules

Transformation rules bridge the gap between the extracted syntactic sentence information and the targeted OOAM. Each transformation rule describes how a combination of words of identified grammatical roles can be transformed into classes, associations, attributes, and methods.

The transformation rules Dowser uses were derived from Juristo *et al.*'s grammar [17], the OO methods literature [27][29], and conducted experiments. In total, Dowser uses 13 transformation rules. The five most frequently used are the following:

1. The most frequent applicable rule is the above described rule. If a parsed sentence contains a subject and an object link, then create two classes with a directed association named after the verb.
2. Aggregations are an important notion in UML class diagrams. One rule for extracting aggregations is similar to the first rule. The major difference is the verb. This rule is applicable only if the parsed sentence contains a subject and an object link and the verb stem is one of have, possess, contain, or include. In that case, the object is aggregated to the subject.
3. Sometimes, a subclause describes a system action without the need of an object, particularly, if the system reacts to a given event, e.g., If the user presses the button, the elevator moves.. An event clause starts with an if or when. If Dowser detects an event clause, and the main clause has only a subject link, then a class from the subject link noun is created and the verb is added to the new class as a method.
4. A genitive attribute indicates two classes with an aggregation, e.g., The system stores the name of the customer. or The system stores the customer's

1. "Uniguity" means "lack of ambiguity'.

name.. If Dowser detects a genitive, it creates two classes with one linking aggregation. For either example, Dowser would create a class customer, and it would aggregate the class name to the class customer; name could have been modeled as an attribute. However, other sentences in the specification would add methods to the class `name` later. Therefore, with the syntactic information of only one sentence, it cannot be decided if name is an attribute or an aggregated class. This rule needs to be constrained by semantic information. For example, Dowser should not apply this rule to the sentence The user enters the amount of money.. Although amount is a noun, the class amount is not desired in this case.

5. Although active clauses are preferred in NL SRSs, passive clauses are still needed. They are used to describe relations and states, e.g. as in A husband is married to his wife.. From this sentence, two classes are created, from the subject noun and the noun of the prepositional phrase. The passive verb and the connecting word `to` link the prepositional phrase described with the association.

Dowser applies two post-processing rules after it executes all possible transformation rules.

The first post-processing rule converts all classes that are aggregated to another class into attributes of that other class. Only a class that lacks any attribute, method, or incoming or outgoing association is transformed. For example, one rule described above extracts two classes and one aggregation from the sentence The system stores the name of the customer.. The rule creates a class name and a class customer. However, the class name has probably no method or association. Therefore, if a class contains no method after all rules have been applied, it is transformed into an attribute of the class customer.

The second post-processing rule removes the class system from the OOAM, since all other classes together form the system; system is not a class, because it cannot be a subpart of itself.

The full set of rules, a user's manual for Dowser, and other details may be found at the Web site, *http://www.cc.gatech.edu/projects/dowser/* .

3.4 Domain-Specific Terms

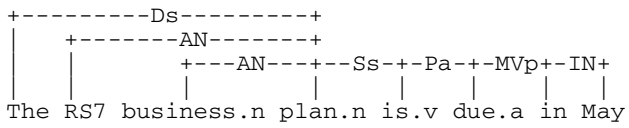
In order to build the correct classes into the OOAM of a NL SRS, our approach has to be able to detect DSTs. If Dowser were to extract only the concept button from elevator button, Dowser would be identifying an incorrect term.

To achieve high DST recall, the parser could access a special domain data dictionary. However, for each of most domains, such a data dictionary does not exist. Software is built for financial, medical, technical, and other domains. Creating a domain dictionary for each of these rich domains is complex and difficult. Additionally, the customer of a software application might prefer to use her own terms for describing her product. A product could have arbitrarily chosen names, such as DNAComp07. Therefore, even if a domain dictionary exists for a NL SRS, DST detection remains a challenge.

The link types of the link grammar parse can be used to identify DSTs. The typical DST happens to be built from an attributive noun or a proper noun. Therefore, in a link

grammar parse, the AN link, denoting an attributive noun, and the G link, denoting a proper noun, help to identify DSTs.

Consider the link grammar parse of the sentence The RS7 business plan is due in May.:



Thus, RS7 business plan is a domain-specific term.

A parser typically has problems parsing words that are not in its internal dictionary. However, the S&T link grammar parser has a guessing mode, in which it can guess the syntactic role of an unknown term. Therefore, it is often able to guess the syntactic role of an unknown term, improving its DST recall.

Since DST detection is essential for transforming a NL SRS into an OOAM, we conducted a study, described in Section 4.2, about the recall and precision of our approach.

3.5 Diagramming OOAMs

The previous steps are able to create a textual OOAM. However, it is easier for a human to understand a graphical OOAM than a textual OOAM. Using the approach described by Spinellis [31], an extracted textual OOAM is diagrammed. The tool *UMLGraph* [32] transforms a textual description into a *dot file*, which the tool *Graphviz* [13] can transform into any of some popular graphic formats, such as JPEG, GIF, or PNG.

3.6 Interpretation of OOAM

In the last step of our approach, a human analyst checks the created diagram for ambiguities.

Some ideas that a human analyst can use to find defects in an OOAM are:

- An association is a hint for possible ambiguities. For example, suppose that each of two different classes sends a message to the same target class. The analyst should check that the two different classes actually are to communicate with the same target class. If a motion sensor activates one type of display, and a smoke detector activates another type of display, then the class diagram should reflect this situation with two different display classes.
- Each class should reflect one and only one concept. For example, the analyst should check that book and textbook are really two different classes when Dowser does not create a generalization of these two classes.
- If a class has an attribute, but the attribute is not of a primitive type, such as string or number, then the definition of the attribute might be missing in the original text. After a definition is added, the attribute should be represented by its own class.
- If a class has no association, then the class might be underspecified, as there are no relations or interactions between the class and other classes.

3.7 Limitations of Method

Observe that the OOAM is a model of only static relationships among the concepts mentioned in the parsed NL SRS. We have not attempted to apply our approach to modeling behavior.

4 Studies

This section describes the case studies in which we evaluated the effectiveness of our approach in helping an analyst to identify ambiguities, inconsistencies, and underspecifications in a NL SRS and in which we evaluated Dowser's effectiveness at DST identification.

4.1 Elevator Case Study

To evaluate the effectiveness of our approach, we implemented Dowser and applied it to an example NL SRS that we call "the ESD". The ESD describes the control software for an elevator system [16]. The ESD was chosen, because it could be the NL SRS of a real industrial system. At the same time, the ESD is short enough to completely described in this paper. Moreover, the ESD happens to contain enough defects that it illustrates the defect types that can be revealed with the help of Dowser.

The original ESD was:

An n elevator system is to be installed in a building with m floors. The elevators and the control mechanism are supplied by a manufacturer The internal mechanisms of these are assumed (given) in this problem.

Design the logic to move elevators between floors in the building according to the following rules:

1. Each elevator has a set of buttons, one button for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited (i.e., stopped at) by the elevator.
2. Each floor has two buttons (except ground and top), one to request an up-elevator and one to request a down-elevator. These buttons illuminate when pressed. The buttons are cancelled when an elevator visits the floor and is either travelling the desired direction, or visiting a floor with no requests outstanding. In the latter case, if both floor request buttons are illuminated, only one should be cancelled. The algorithm used to decide which to serve first should minimize the waiting time for both requests.
3. When an elevator has no requests to service, it should remain at its final destination with its doors closed and await further requests (or model a "holding" floor).
4. All requests for elevators from floors must be serviced eventually, with all floors given equal priority.
5. All requests for floors within elevators must be serviced eventually, with floors being serviced sequentially in the direction of travel.

First, we applied Dowser tool to the original unmodified ESD, which was not written according to any constraining grammar. Since the transformation rules have been created assuming that the analyzed text conforms to a constraining grammar, applying Dowser to the original ESD resulted in a diagram with only five classes such as these and set. None of these classes describes any domain concepts of the ESD.

To successfully apply Dowser to the ESD, the ESD had to be rewritten sentence-by-sentence to conform to the constraining grammar. No information was added or removed from the original ESD during the rewriting. Therefore, the rewriting did not introduce any new defects, which would have adulterated the results of the case study.

The rewritten ESD is:

An n elevator system is to be installed in a building with m floors.

1. Each elevator has buttons. Each elevator has one button for each floor. When a user presses a button, the elevator illuminates the button and the elevator visits the corresponding floor. When the elevator visits a floor, the elevator cancels the corresponding illumination.
2. Each floor has two buttons. (except ground and top). If the user presses the up-button, an up-elevator is requested. If the user presses the down-button, a down-elevator is requested. If the user presses a button, this button becomes illuminated. When an elevator visits a floor, the elevator cancels the corresponding illumination of the button in the desired direction. The system minimizes the waiting time.
3. When an elevator has not to service any requests, the elevator remains at its final destination and the doors of the elevator are closed. The elevator then awaits further requests.
4. The elevators service all requests from floors with equal priority eventually.
5. If a user presses a button within the elevator, the elevator services this request eventually in the direction of travel.

Applying Dowser to the new ESD resulted in the diagram of Figure 2. Dowser created an OOAM containing 14 partially connected classes with attributes and methods.

The graphically rendered OOAM does not reveal defects on its own. By applying the guidelines described in Section 3.6, we identified four conceptual defects in the OOAM, which could be traced back to the original ESD.

1. The diagram shows classes up-elevator and down-elevator. Neither class has any connection to any other class, and neither has any association to the class elevator. Furthermore, the up-elevator has an attribute requested, while elevator serves a request indicates that each of up-elevator and down-elevator is a specialization of elevator. All of this information indicates that neither concept, up-elevator nor down-elevator, is defined enough in the original ESD.
2. The class door in the diagram contains the attribute closed. However, the class has no method to manipulate this state. If closing and opening the door are within the scope of the system, then it is clear that the concept door is not defined enough in the original ESD.

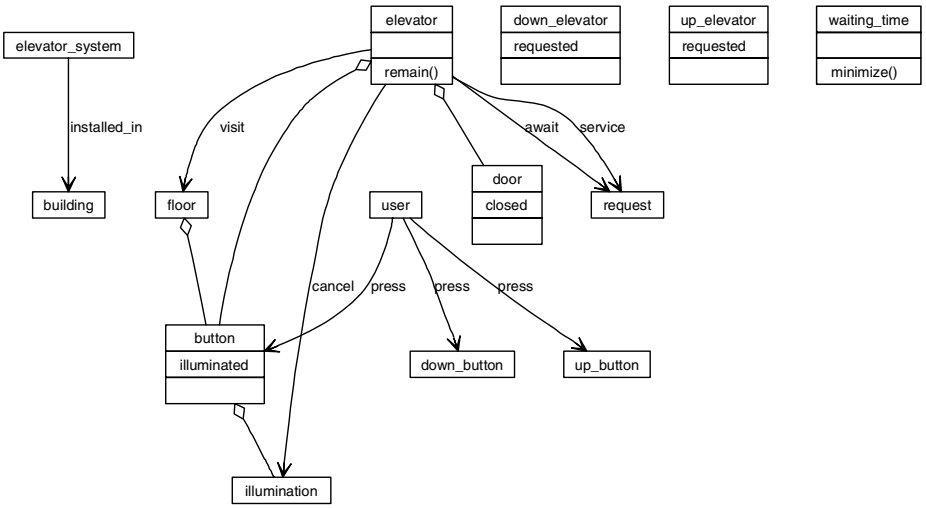


Figure 2. OOAM of the ESD

3. In the ESD, each of the floor and the elevator has buttons. Therefore, each of the class elevator and the class floor should have an aggregated class button in the OOAM. However, the diagram indicates that both have the same type of button. Since a button in an elevator and a button in a floor have different behaviors, it is unlikely that the same class describes both types of buttons. Generalizing both types to a single class button could therefore lead to misinterpretations. Defining concepts elevator button and floor button would resolve this ambiguity and enhance the clarity of the ESD.
4. Each of the classes up-button and down-button is connected to only the class user in the OOAM. Since a user is an actor in the system, the diagram does not clarify where button belongs. The location can be derived from the ESD, because button is mentioned in the paragraph that mentions floor. However, it should not be necessary to use this fact. Therefore, each concept should be specified in more detail in the ESD to reduce the ambiguity in the specification.

This case study shows how Dowser can help an analyst identify defects in a NL SRS. If a constraining grammar is used to write a NL SRS, our approach can help detect ambiguities, inconsistencies, and underspecifications.

4.2 DST Detection Quality

The case study in Section 4.1 shows the importance of detecting DSTs. For the ESD, Dowser needed to be able to detect DSTs such as floor button, elevator button, or down elevator. If Dowser were to extract only the terms button and elevator from the ESD, it would create wrong classes and associations.

Section 3.4 explains how Dowser relies on syntactic information to identify DSTs. To measure Dowser's DST detection capability, we conducted a separate study. To measure Dowser's DST detection, the metrics *recall* and *precision* were calculated of

Dowser's extraction of DSTs from a user's manual. These metrics are used to evaluate the success of many NLP tools [14] :

- *Recall* measures how well a tool identifies desired pieces of information in a source:

$$Recall = \frac{I_{correct}}{I_{total}}$$

  Here $I_{correct}$ is the number of correctly identified desired pieces of information in the source text and I_{total} is the total number of desired pieces of information in the source text.

- *Precision* measures how accurately a tool identifies desired pieces of information in a source:

$$Precision = \frac{I_{correct}}{I_{correct} + I_{incorrect}}$$

  Here $I_{correct}$ is as for *Recall* and $I_{incorrect}$ is the number of incorrectly identified desired pieces of information in the source text.

We chose the *intro man page* of the *Cygwin environment* with which to measure recall and precision of Dowser's DST detection. A manual page seems to be a suitable experiment source, because it is a technical document with a large number of DSTs.

The steps of the experiments are as follows.

1. We manually identified every DST, a noun or a compound noun, from the *intro man page*. The *intro man page* contains 52 terms like Cygwin, Linux-like environment, Linux API emulation layer, and POSIX/SUSv2. The 52 terms consists of 33 single-noun terms and 19 compound-noun terms.
2. For the first experiment, the link grammar parser extracted every term out of the *intro man page* without the capability of extracting compound-noun terms. It recognized 31 of the single-noun terms and none of the compound-noun terms. Therefore, it reached a recall value of 59.6% for all terms and of 93.9% for single-noun terms.
3. For the second experiment, compound-noun term detection was added to the link grammar parser. After this, the tool recognized 10 compound-noun terms. As the single-noun terms detection rate stayed the same, the tool recognized 41 terms. Therefore, it reached a recall value of 78.8%.

Afterwards, the undetected terms were examined. It turned out that five terms were undetected because they appeared in grammatically obscure or wrong parts in the sentence. Correcting these sentence, increased the detected terms to 46 and the recall value to 88.46%.

The five not identified terms were (1) case-insensitive file system; (2) intro man page; (3) Linux look and feel; (4) Red Hat, Inc.; and (5) User's Guide. The term Red Hat, Inc. is not recognized because of the comma, User's Guide cannot be detected syntactically, because if every genitive were part of a term, it would lead to an over-generation of terms. Linux look and feel is not recognized because of the con-

junction *and* in the term. Case-insensitive file system and intro man page can be only partially detected, because case-insensitive is an adjective, which is only sometimes part of a term. Another example demonstrates the difficulties caused by adjectives. In readable manual, only manual is a term in most cases. Using every adjective as part of the term would lead to an overgeneration of terms. The term intro man page is not recognized because the link grammar parser guesses that intro is an adjective. However, if it is planned that case-insensitive file system is a concept within a SRS, then writing it with initial upper-case letters would allow the link grammar parser to detect it as a proper noun, and thus as a DST.

Dowser extracted seven wrong terms, since it created wrong terminology for the incompletely detected terms, e.g., it extracted the term man page instead of intro man page. Overall, Dowser reached a precision value of 86.79% on the *intro man page*.

The DST detection experiment shows that using only syntactic information from the link grammar parser allows a fairly high DST detection rate.

5 Discussion and Conclusion

Dowser's effectiveness in helping a human analyst to identify ambiguity, inconsistency, and underspecification in a NL SRS and its recall and precision in identifying DSTs in a user's manual are not bad considering that Dowser was built mostly out of existing software.

Of course, Dowser's lack of perfection, particularly in the construction of an OOAM and in the DST recall, says that Dowser can be used as only one of an array of approaches and tools for identifying DSTs and for detecting ambiguity, inconsistency, and underspecification in NL SRSs. However, because of the inherent difficulty of these tasks for humans, every little bit helps!

One drawback of the approach is that for best results, the input should be written in the constrained language. If the actual input is not written in the constrained language, it must be rewritten. This rewriting necessity might be considered a reason not to use Dowser. However, one could argue that the rewriting is part of the whole process of eliminating ambiguity, which ultimately the human carries out.

5.1 Future Work

The lack of perfection says that more work is needed:

- How does Dowser perform on larger, industrial-strength NL SRSs? Answering this question would help to explore the problem space and to find new unsolved research questions.
- The current Dowser cannot resolve anaphora. An anaphor is a linguistic unit, such as a pronoun, that refers to a previous unit. In The customer can buy text books and return them., them is an example of an anaphor, which must be resolved to text books. While Dowser can identify anaphora, it cannot resolve them. A simple solution would be to have Dowser flag all anaphora in its input text, so a human analyst could change each to its resolution.

- DST identification can be improved. As mentioned above, syntactic information is not sufficient to detect all the DSTs within a document. Therefore, frequency analysis or baseline text analysis [20] might improve DST identification.
- Additional semantic knowledge could improve the capability of Dowser. For example, the *WordNet* lexicon contains information about hypernyms, which can indicate superclasses, and meronyms, which can indicate aggregations. This information could be used to supply missing links in an OOAM. However, although *WordNet* is a large online lexicon, it lacks DSTs and therefore might be only a little help. Extending Dowser's dictionary with DSTs could reduce this problem.
- The current Dowser is not applicable to a NL SRS that has a functional language style, i.e., with sentences such as, The system must provide the functionality of.... Handling such sentences would require a different grammar. Future work could examine which grammar is the most suitable for class extraction.
- The UML offers a set of different diagram types. NLP could be used to create sequence, state, or other diagrams. For example, Juristo *et al.* [17] developed also a controlled grammar for specifying dynamic behavior.
- Other work has found different sources of ambiguities in NL SRS. Since there seems not to be a single perfect approach, different approaches (e.g. [35][8][21]) could be integrated into a single framework for validating NL SRSs. This integration could lead to a new method of developing NL SRSs. Indeed, this sort of integration would deal with any syntactic ambiguities found by the parser that is used.

References

[1] IEEE Standard for software reviews and audits, Soft. Eng. Tech. Comm. of the IEEE Computer Society, 1989, IEEE Std 1028–1988.

[2] R. J. Abbott. Program design by informal English descriptions. Communication of the ACM, 26(11): 882–894, 1983.

[3] A. Bucchiarone, S. Gnesi, and P. Pierini. Quality Analysis of NL Requirements: An Industrial Case Study, Proceedings of the Thirteenth IEEE International Conference on Requirements Engineering (RE'05), pp. 390–394, Paris, France, 29 August–2 September, 2005.

[4] S. Delisle, D. Barker and K. Biskri. Object-oriented analysis: Getting help from robust computational linguistic tools. Application of Natural Language to Information Systems, G. Friedl, H.C. Mayr, eds., Oesterreichische Computer Gesellschaft, pp. 167–172, 1999.

[5] D. M. Berry, K. Daudjee, J. Dong, I. Fainchtein, M.A. Nelson, and T. Nelson. User's Manual as a Requirements Specification: Case Studies. Requirements Engineering Journal, 9(1): 67–82, February 2004.

[6] D.M. Berry and E. Kamsties. Ambiguity in Requirements Specification. Perspectives on Requirements Engineering, J.C.S.P. Leite and J. Doorn, eds., Boston, MA, Kluwer, pp. 7–44, 2004.

[7] R. Clayton, S. Rugaber, and L. Wills. Dowsing: A Tools Framework for Domain-Oriented Browsing Software Artifacts. Proceedings of the 1998 Automated Software Engineering Conference, pp. 204–207, Honolulu, HI, USA, May 1998.

[8] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami. The Linguistic Approach to the Natural Language Requirements, Quality: Benefits of the use of an Automatic Tool, Proceedings of the Twenty-Sixth Annual IEEE Computer Society –NASA GSFC Software Engineering Workshop, pp. 97–105, Greenbelt, MA, 27–29 November, 2001.

[9] N. Fuchs and R. Schwitter. Attempto controlled English (ACE). The First International Workshop On Controlled Language Applications (CLAW), Belgium, 1996.

[10] N. E. Fuchs, U. Schwertel, and R. Schwitter. Attempto Controlled English (ACE) Language Manual, Version 3.0. Technical Report 99.03, Department of Computer Science, University of Zurich, August 1999.

[11] D. Gause and G.Weinberg. Exploring Requirements: Quality before Design, New York, NY, Dorset House, 1989.

[12] V. Gervasi and B. Nuseibeh. Lightweight validation of natural language requirements. Softw., Pract. Exper., 32(2): 113–133, 2002.

[13] Graphviz—Graph Visualization Software home page, accessed 8 August 2007, http://www.graphviz.org/Credits.php.

[14] R. Grishman. Information extraction: Techniques and challenges. SCIE'97: International Summer School on Information Extraction, pp. 10–27, London, UK, 1997. Springer-Verlag.

[15] H. M. Harmain and R. J. Gaizauskas. CM-builder: A natural language-based CASE tool for object-oriented analysis. Autom. Softw. Eng., 10(2): 157–181, 2003.

[16] M. Heimdahl. An example: The lift (elevator) problem. http://www-users.cs.umn.edu/heimdahl/formalmodels/elevator.htm, accessed on 14.12.2005.

[17] N. Juristo, A. M. Moreno, and M. Lopez. How to use linguistic instruments for object-oriented analysis. IEEE Softw., 17(3): 80–89, 2000.

[18] L. Kof. Natural Language Processing for Requirements Engineering: Applicability to Large Requirements Documents. Automated Software Engineering, A. Russo, A. Garcez, and T. Menzies, eds., Proceedings of the Workshops, Linz, Austria, September 21, 2004.

[19] N. Kiyavitskaya, N. Zeni, L. Mich, and D. M. Berry. Requirements for Tools for Ambiguity Identification and Measurement in Natural Language Requirements Specifications, Technical Report, School of Computer Science, University of Waterloo, Waterloo, ON, Canada, 2007, http://se.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/KZMB2007AmbTR.pdf.

[20] R. Lecoeuche. Finding Comparatively Important Concepts between Texts. Proceedings of the 15th IEEE international Conference on Automated Software Engineering, Grenoble, France, 2000.

[21] L. Mich. On the Use of Ambiguity Measures in Requirements Analysis, Proceedings of the Sixth International Conference on Applications of Natural Language to Information Systems (NLDB), A. Moreno and R. van de Riet, eds., pp. 143–152, Madrid, Spain, 28–29 June, 2001.

[22] L. Mich, M. Franch, and L. N. Inverardi. Market research for requirements analysis using linguistic tools. Requir. Eng., 9(2): 151–151, 2004.

[23] L. Mich and R. Garigliano. NL-OOPS: A requirements analysis tool based on natural language processing. Proceedings of Third International Conference on Data Mining Methods and Databases for Engineering, Bologna, Italy, 2002.

[24] L. Mich and R. Garigliano. Ambiguity measures in requirements engineering. Proceedings of International Conference on Software–Theory and Practice (ICS2000), Sixteenth IFIP World Computer Congress, Beijing, China, pp. 39–48, August 2000.

[25] G. A. Miller, C. Felbaum, *et al*. WordNet Web Site. Princeton University, Princeton, NJ, USA, accessed 12 March 2006, http://wordnet.princeton.edu/

[26] D. Mollá, R. Schwitter, F. Rinaldi, J. Dowdall, and M. Hess. NLP for answer extraction in technical domains. Proceedings of the EACL 2003, Workshop on Natural Language Processing (NLP) for Questions Answering, Budapest, Hungary, pp. 5–12, April 14, 2003.

[27] S. Nanduri and S. Rugaber. Requirements validation via automated natural language parsing, Journal of Management Information Systems 12(3): 9–19, Winter 1995–96.

[28] K. Ryan. The role of natural language in requirements engineering. Proceedings of the International Symposium on Requirements Engineering, pp. 240–242, San Diego, CA, January 1993.

[29] J. Rumbaugh. Object-Oriented Modeling and Design. Englewood-Cliffs, NJ, Prentice Hall, 1991.

[30] D. D. Sleator and D. Temperley. Parsing English with a link grammar. Proceedings of the Third International Workshop on Parsing Technologies, http://www.link.cs.cmu.edu/link/papers/index.html, 1993.

[31] D. Spinellis. On the declarative specification of models. IEEE Software, 20(2): 94–96, March/April 2003.

[32] D. Spinellis. UMLGraph - Declarative Drawing of UML Diagrams. http://www.spinellis.gr/sw/umlgraph/, accessed 8 August 2007.

[33] R. Sutcliffe and A. McElligott. Using the link parser of Sleator and Temperley to analyse a software manual corpus. Industrial Parsing of Software Manuals, pp. 89–102, 1995.

[34] D. Temperley, D. Sleator, and J. Lafferty. Link Grammar Home Page, accessed 1 August 2007, http://www.link.cs.cmu.edu/link/.

[35] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. Automated Analysis of Requirement Specifications, Proceedings of the Nineteenth International Conference on Software Engineering (ICSE 97), pp. 161–171, Boston, MA, USA, 17–23 May, 1997.

Environment Models for Specifying Functional and Non-Functional Requirements for Reactive Systems

(Position paper)

Mikhail Auguston

Naval Postgraduate School, Computer Science Department,
Monterey, California 93943-5118, USA
maugusto@nps.edu

Abstract. A formalism is suggested for specifying environment behavior models for software test scenario generation based on attributed event grammars. The environment model may contain descriptions of the events triggered by the software outputs and of the hazardous states in which the system could arrive, thus providing a framework for specifying properties of software behavior within the given environment. The behavior of the system can be rendered as an event set with two partial ordering relations: precedence and inclusion (event trace). This formalism may be used as a basis for automation tools for test generation, test result monitoring and verification, for experiments to gather statistics about software safety, and for evaluating of dependencies of system's behavior on environment parameters. The monitoring activities can be implemented within a uniform framework as computations over event traces.

Keywords: environment models, reactive systems, requirements specification and verification, testing and safety assessment automation, event traces.

1 Introduction

Reactive and real-time systems are at the core of many safety-critical software applications. In [1] an approach to testing automation for reactive and real-time software systems based on attributed event grammars (AEG) has been introduced. The main idea is to specify the environment behavior model as a set of events that control the inputs for the system under the test (SUT) and that may adjust the behavior depending on the outputs provided by the SUT (adaptive testing [5]). Figure 1 outlines the major steps in the testing process based on AEG.

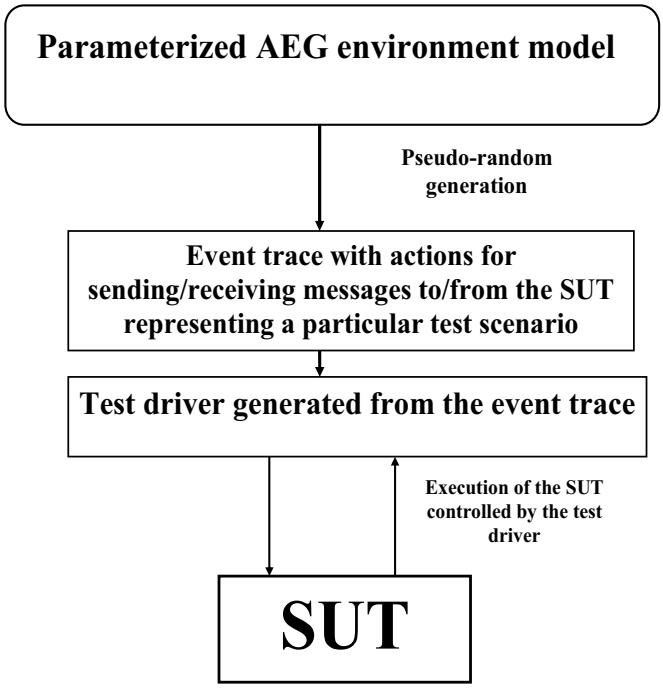


Figure 1. Testing automation framework based on attributed event grammars as environment models.

2. The Environment Model

The notion of event is central for our approach. An **event** is any detectable action in the environment that could be relevant to the operation of the SUT. A keyboard button pressed by the user, a group of alarm sensors triggered by an intruder, a particular stage of a chemical reaction monitored by the system, and the detection of an enemy missile are examples of events. In this approach an event usually is a time interval, and has a beginning, an end, and a duration. An event has **attributes**, such as type and timing attributes.

There are two basic relations defined for events: **precedence** (PRECEDES) and **inclusion** (IN). Two events may be ordered, or one event may appear inside another event. The behavior of the environment can be represented as a set of events with these two basic relations defined for them (**event trace**). Usually event traces have a certain structure (or constraints) in a given environment. The basic relations define a partial order of events. Two events are not necessarily ordered; that is, they can happen concurrently.

The structure of possible event traces can be specified by **event grammar**. Here identifiers stand for event types, sequence denotes precedence of events; (…|…) denotes alternative; * means repetition zero or more times of ordered events; {a, b} denotes a set of two events a and b without an ordering relation between them; and {…}* denotes a set of zero or more events without an ordering relation between them. The rule *A::= B C* means that an event of the type *A* contains (IN relation) ordered events of types *B* and *C* correspondingly (PRECEDES relation).

Example 1.

```
OfficeAlarmSystem::= {DoorMonitoring,
                      WindowMonitoring }
```

The `OfficeAlarmSystem` behavior is a set of two concurrent monitoring threads.

```
DoorMonitoring::= DoorSensor *
```

The `DoorMonitoring` is a composite event, which contains a sequence of ordered events of the type `DoorSensor`.

```
WindowMonitoring::= WindowSensor *

DoorSensor::= ( DoorClosed | DoorAlarm )
```

The `DoorSensor` event may contain one of two possible alternatives.

```
WindowSensor::= ( WindowClosed | WindowAlarm )
```

This event grammar defines a set of possible event traces – a model of a certain environment. The purpose is to use it as a production grammar for random event trace generation by traversing grammar rules and making random selections of alternatives and numbers of repetitions.

2.1 Event Attributes

An event may have attributes and actions associated with it. Each event type may have a different attribute set. Event grammar rules can be decorated with attribute evaluation rules. The */action/* is performed immediately after the preceding event is completed. Events usually have timing attributes like *begin_time*, *end_time*, and *duration*. Some of those attributes can be defined in the grammar by appropriate actions, while others may be calculated by appropriate default rules. Attributes can be either inherited or synthesized; we assume that all attribute evaluation is accomplished in a single pass, and the event grammar is traversed top-down, left-to-right for producing a particular event trace. The interface with the SUT can be specified by an action that sends input values to the SUT or listens for a message sent

by the SUT. This may be a subroutine in a common programming language like C or Java that hides the necessary wrapping code.

Example 2.

An (over)simplified environment model for a missile defense system that tracks radar sensors and at certain moment sends a command to proceed with an interception.

```
Attack::= { Missile_launch }* (=N)
```

The `Attack` event contains N concurrent `Missile_launch` events.

```
Missile_launch::=
   Boost_stage
   Middle_stage
   WHEN(Middle_stage.completed)     Boom
```

The `Boom` event (which happens if the interception attempts have failed) represents an environment event, which the SUT should try to avoid. It represents a "hazard state" in which the system may arrive.

```
Middle_stage::=
  / Middle_stage.completed := True/

   ( move

     CATCH  SUT_launch_interception(hit_coordinates)
         WHEN (hit_coordinates ==
                  Middle_stage.coordinates )
         [ p(p1) interception
           / Middle_stage.completed := False;
             send_success( Middle_stage .coordinates);
             BREAK; /   ]

    )* (<=M, EVERY 50 msec)
```

The sequence of *move* events within `Middle_stage` event may be interrupted by receiving of an external event from the SUT. This will suspend the *move* event sequence and will try to continue with event *interception* (with probability p1), which simulates the missile interception event triggered by the SUT, followed by sending notification to the SUT about the successful interception and the `BREAK` command, which will terminate the `move` event iteration. If the `interception` event was not generated, the `move` sequence will resume. This model allows several interception attempts during the same missile launch event.

```
move ::=
 /adjust( ENCLOSING Middle_stage.coordinates);
  send_radar_signal(ENCLOSING Middle_stage.coordinates);/
```

This rule provides attribute calculations and sends an input to the SUT. The `ENCLOSING` construct provides access to the attributes of parent event. In general, external events (i.e., events generated by the SUT) may be broadcasted to several event listeners in the AEG, or may be exclusive and consumed by just one of the listeners. These interface details are encapsulated in listener subroutines like `SUT_launch_interception()` where the parameter `hit_coordinates` is passed by reference.

3. Behavior Properties Specification

The next problem to be addressed after the system behavior model is set up is the formalism specifying properties of the behavior. As a unifying framework we came up with the concept of a computation over the event trace. This approach implies the design of a special programming language for computations over the event traces. In [2], [3], [4], a language FORMAN, based on functional paradigm and the use of event patterns and aggregate operations over events, is suggested.

Event patterns describe the structure of events with possible context conditions. Execution paths can be described by path expressions over events. This makes it possible to write assertions not only about pre-conditions and post-conditions at event trace points, but also about data flows in the entire trace.

The subroutine calls for inputs in the SUT and for catching outputs from the SUT can be considered also as events with obvious precedence and inclusion relations with the rest of event trace. The parameter values at the beginning and the end of those events are specific attributes that provide the opportunity to write assertions about system input/output values at different points in the execution history.

3.1 The Language for Computations over Event Traces

FORMAN is a high-level specification language for expressing intended behavior or known types of error conditions when debugging or testing programs. FORMAN supplies a means for writing assertions about events and event sequences and sets. Monitoring activities can be implemented as computations over event traces. Typical examples of monitoring include:

- Assertion checking (test oracles)
- Debugging queries

- Profiles
- Performance measurements
- Behaviour visualization

The following provides an outline of the FORMAN constructs. More details are available in [2], [3], [4]. The environment model from Example 2 will be used as a background for further examples.

Event patterns

```
x: Middle_stage & x.Value_at_end(completed)== False
```

This pattern matches an event of the type `Middle_stage` if and only if the value of the `completed` attribute at the end of this event is `False`.

List of events

Assuming that `m` is an event of the type `Middle_stage`.

```
[ move FROM m ]
```

This creates a list of `move` events from the enclosing even m preserving the precedence relation between them.

List of values

Assuming that `m` is an event of the type `Middle_stage`.

```
[ x: move FROM m APPLY x.Value_at_end( m.coordinates ) ]
```

This creates a list of values of `coordinates` attribute of the enclosing `Middle_stage` event `m` taken at the end of each `move` event inside `m`. Note that the value of `m.coordinates` may change after each `move` event.

Aggregate operations

Assuming that `m` is an event of the type `Middle_stage`.

```
OR/[ x: SUT_launch_interception FROM m
    APPLY x.param[1] == x.Value_at_end( m.coordinates )]
```

This expression yields a Boolean value depending on whether there is at least one instance x of `SUT_launch_interception` event inside m that yields True for the expression `x.param[1] == x.Value_at_end(m.coordinates)`. The

`x.param[1]` denotes the value of the first actual parameter of the subroutine `SUT_launch_interception` call. This aggregate operation can be abbreviated as:

```
EXISTS x: SUT_launch_interception FROM m

    ( x.param[1] == x.Value_at_end( m.coordinates ))
```

In a similar way, FOREACH quantifier can be introduced as an abbreviation for the AND/ aggregate operation.

Generic requirements for the SUT behaviour within the given environment can be specified in FORMAN. The following examples illustrate this.

Example 3.

The requirements for the SUT may include for example the following: "There is at least one interception attempt for each `Missile_launch` event within the `Attack` event."

```
FOREACH x: Missile_launch FROM Attack

      EXISTS y: SUT_launch_interception FROM x
```

Example 4.

The first interception attempt should happen no later than 1 sec after the beginning of the `Missile_launch` event.

```
FOREACH x: Missile_launch FROM Attack

      EXISTS y: SUT_launch_interception FROM x

            y.begin_time - x.begin_time < 1 sec
```

Example 5.

There should not be unintercepted missile launches.

```
CARD/[ Boom FROM Attack]  == 0
```

The examples of FORMAN expressions above represent computations over the event traces and can be performed during the test run or after it based on a log file collected during the test run. This supports the requirement tracing as a part of testing process.

This framework provides means for expressing quantifiers over events and ordering and inclusion relations for events and is comparable with the expressive power of other specification formalisms for behavior specification, such as temporal logic and abstract event traces [6], [7].

4. Conclusions and Future Work

Traditionally, reactive systems and their environments are modeled with some kind of finite state machine, like Statecharts or timing automata. For the purposes of scenario (and corresponding test case) generation, the AEG approach may have several useful features, in particular:

- It is based on a precise behavior model in terms of an event trace with precedence and inclusion relations, well suited to capture hierarchical and concurrent behaviors. Since an event may be shared by other events, the model can represent synchronization events as well.
- The control structure suggested by the event grammar notation (sequence, alternative, iteration, concurrent event set) and the top-down, left-to-right order of traversal seems to be intuitive and close to the traditional imperative programming style, hence facilitating the design of models.
- Data flow of attributes is integrated with the control flow (i.e., event trace), and AEG notation provides means for ease of navigation within the derivation tree (e.g., the ENCLOSING event construct for referencing parent event attributes on any distance in the derivation tree).
- The probabilities for alternatives or number of iterations may be attached to meaningful events in the model and are more intuitive and less numerous than in Markov models based on finite state machines. This provides for a natural definition of functional profiles for scenario generation.

The main advantages of the suggested approach may be summarized as follows.

- Environment models specified by attributed event grammars provide for automated generation of a large number of pseudo-random (but satisfying the constraints) test drivers. This feature provides for gathering of statistical data for safety assessment experiments.
- All attribute values that do not depend on the SUT output can be calculated at the generation time. As a result, the generated test driver contains only actions that should be postponed to the run time (like sending inputs to the SUT and listening to the SUT outputs), has a low overhead, and could be used for real-time test drivers.
- As any notation based on formal grammars, AEG is well structured, hierarchical, and scalable.

- The environment model may contain events that represent hazardous states of the environment. Experiments with the SUT embedded in the environment model ("software-in-the-loop") provide a constructive method for quantitative and qualitative assessment of software.
- Different environment models for different purposes can be designed, such as for testing extreme scenarios by increasing probability or number of certain events, or for load testing. The same safety assessment methodology as described above may be applied for these special cases as well. The environment model itself is an asset and can be reused.
- It addresses the regression testing problem – generated test drivers can be saved and reused. We expect that environment models will be changed relatively seldom unless serious requirement errors are discovered during testing.
- Event traces generated from the AEG model represent examples of SUT interaction with the environment, and are in fact use cases, which could be useful for requirements specification and other prototyping tasks.
- The FORMAN expressions defining computations over event traces provide a uniform framework to specify functional and not-functional properties of the system under test, and can be used for test monitoring and result verification.

Essentially the framework provides a constructive way for tracing the conformance with requirements specifications during the test run.

This work is under progress. In order to automate software assessment we are developing a tool kit that will integrate the AEG test generator with the combinatorial testing suite tool CTS developed by Alan Hartman and Leonid Raskin [8], and the test result verification based on computations over event traces.

References

[1] M.Auguston, B.Michael, M.Shing, "Environment Behavior Models for Automation of Testing and Assessment of System Safety," Information and Software Technology, Elsevier, Volume 48, Issue 10, October 2006, pp. 971-980

[2] M. Auguston, "FORMAN -- A Program Formal Annotation Language," Proceedings of the 5th Israel Conference on Computer Systems and Software Engineering, Gerclia, May 1991, IEEE Computer Society Press, pp.149-154.

[3] M. Auguston, "Program Behavior Model Based on Event Grammar and Its Application for Debugging Automation," in Proceedings of the 2nd International Workshop on Automated and Algorithmic Debugging, Saint-Malo, France, May 1995.

[4] P. Fritzson, M. Auguston, N. Shahmehri, "Using Assertions in Declarative and Operational Models for Automated Debugging," The Journal of Systems and Software 25, 1994, pp. 223-239.

[5] R. M.Hierons, H.Ural, “Concerning the Ordering of Adaptive Test Sequences,” in Proc. 23rd IFIP Int. Conf. on Formal Techniques for Networked and Distributed Systems, Berlin, Germany, Sept. 2003, Berlin: Springer, Lecture Notes in Computer Science, Vol. 2767, pp. 289-302.

[6] Zohar Manna and Amir Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification,. Springer-Verlag, 1992.

[7] A.Mazurkiewicz, Trace Theory, in W. Brauer et al., editors, Petri Nets, Applications and Relationship to other Models of Concurrency, Vol. 255, Lecture Notes in Computer Science, Springer Verlag, 1987, pp.279-324.

[8] IBM Combinatorial Test Services, http://www.alphaworks.ibm.com/tech/cts

Innovations on Natural Language Document Processing for Requirements Engineering

V. Berzins, C. Martell, Luqi, V. Ivanchenko
Naval Postgraduate School
Monterey CA

1. Introduction

A major challenge in requirements engineering is dealing with changes, especially in the context of systems of systems with correspondingly complex stakeholder communities and critical systems with stringent dependability requirements. Documentation driven development (DDD) is a recently developed approach for addressing these issues that seeks to simultaneously improve agility and dependability via computer assistance centered on a variety of documents [25-27]. The approach is based on a new view of documents as computationally active knowledge bases that support computer aid for many software engineering tasks from requirements engineering to system evolution, which is quite different from the traditional view of documents as passive pieces of paper. Value added comes from automatically materializing views of the documents suitable for supporting different stakeholders and different automated processes, as well as transformations that connect different levels of abstraction and representation.

At the level of requirements engineering, the central problems are related to bridging the gap between stakeholders, who communicate in natural language, and software tools, which depend on a variety of formal representations. A prominent problem is resolving ambiguity, which is typical of natural language and to a somewhat lesser degree the popular informal design notations such as UML. Others include finding implied but unstated requirements, detecting conflicts between needs of different stakeholders, and resolving such conflicts.

Progress on increasing flexibility without damaging reliability depends on computer aid, and in an end-to-end process that includes requirements engineering, this leads to a need for natural language processing that can help bridge the gap between natural stakeholder communication and unambiguous requirements models such as those embodied in the DDD view of documents.

In the 1970's the automatic programming group at MIT headed by Prof. Bill Martin sought to create an end-to-end system that went from user requirement documents to running code for business information systems. The project made progress at the top and bottom levels of this process, but the two ends were never integrated together.

The capabilities of natural language processing (NLP) software and our understanding of requirements engineering (RE) have improved substantially over the past 30 years. This paper re-examines how the current state of NLP can contribute to requirements engineering, how close is it to making a practical impact in the context, and what needs to be improved to enable widespread adoption. We examine the connection between a

hypothetical NLP front end and requirements engineering processes that would follow, and identify some of the differences between generic NLP and domain-specific NLP embedded in a requirements engineering process.

1.1. Challenges of NLP for Requirements Engineering

Requirements engineering is a critical part of the system development process because requirement errors cost roughly 100 times less to correct during requirement engineering than after system delivery [23]. This imposes extreme constraints on the accuracy of NLP that we might use to derive system requirements. However, NLP accuracies are currently in 90%-92% range, at best (see section 2). Therefore NLP must be augmented with other methods for removing residual errors, and accuracy must be greatly improved if it is to be seriously used for RE.

1.2. Why All is Not Yet Lost

NLP in the context of RE should be more tractable than generic NLP, because it has the usual advantages of a domain-specific approach: scope is narrower, more is known about the context, and specialized methods may apply. In particular, much more is known about the intentions of the speaker and the context, such as typical goals and surrounding tasks.

1.3. Overview

Section 2 briefly summarizes recent trends in NLP. Section 3 outlines basic issues in requirements engineering and how they relate to interactions between a NLP front end and system development processes that follow. Both aspects have been simplified to help bridge the gap between the two communities; our advanced apologies to experts in both domains for leaving out some of the subtleties of each area. Section 4 outlines some improvements to NLP that may be possible in the context of RE. Section 5 concludes with an assessment of what should be done to improve likelihood of practical impact in this direction.

2. Summary of recent trends in Natural Language Processing

Natural Language Processing (NLP) is a cross-cutting discipline that includes computer science, linguistics, artificial intelligence and cognitive science, as well as statistics and information theory. The objective of NLP is automated understanding and generation of written natural languages (NL). Challenges of NLP include: the complexity and ambiguity of language constructs; the fact that understanding a natural language often requires representation of one's knowledge about the outside world (tacit knowledge); and the fact that non-linguistic context might also need to be considered, since it often helps to improve the interpretation of speaker intentions. Table 1 provides some concrete examples of the problems just mentioned. On the one hand, research in the young field of NLP still struggles with conceptual difficulties such as context modeling or formalization of speaker intentions [1]. On the other hand, the initial period of excessive optimism in

the field was followed by mature statistical analysis and creation of extensive linguistic resources that have helped foster excellent progress in many NLP domains, e.g., part-of-speech-tagging and parsing.

Table 1. Challenges of NLP

Problem	Examples
1. Ambiguity of word meaning and scope	The word *bank* can refer to either a river bank or a financial institution. Natural languages usually don't specify which word an adjective modifies. For example, in the string "pretty little girls' school"; do the adjectives refer to "girl" or to "school"?
2. Complexity	For determination of grammaticality, it is possible that an exponential number of parse trees might need to be checked.
3. Tacit knowledge and anaphora resolution	The sentences *We gave the monkeys the bananas because they were hungry* and *We gave the monkeys the bananas because they were over-ripe* have the same surface grammatical structure. However, in one of them the word *they* refers to the monkeys, in the other it refers to the bananas: the sentence cannot be understood properly without knowledge of the properties and behavior of monkeys and bananas.
4. Non-linguistic Context	Includes facial expression, gestures, disfluencies, time of the year/day, recent events, etc.

One of the important methodological developments in NLP research was identifying different levels of representation and processing, each with their own set relevant entities, statistical relations, problems and solutions. NLP distinguishes at least four processing levels: lexical, syntactic, semantic, and pragmatic. Each level has its own patterns of ambiguity (see table 2) and corresponding processing methods. As a rule of thumb, the higher the level, the longer the contextual dependencies that have to be taken into account. Importantly, processing at each level is not generally independent. For example, knowing semantics of a sentence may help to disambiguate the part of speech for a particular word.

NLP can be viewed as a sequence of processing steps that starts from a raw text and proceeds through each higher level of representation. Under this approach the output of a lower level is the input for a higher level. Though there are some interdependencies, for simplicity each level is most often considered independently. This assumption greatly facilitates the identification of specific features at each level. It is also important to note that while processing on lexical and syntactical levels is relatively well defined, the higher levels of NLP are not standardized in terms of their objectives or output formats. This is due to the overall complexity of the processing on higher levels and in the extra-linguistic features involved. For example, in order to define pragmatic content for a text one needs to know the intentions of the reader or writer, which typically are not the part of a text. To clarify this point, consider a reader who tries to identify a timeline of events in the text versus another reader who is interested in the text topic. Thus, a simplified vision of NLP is sequential processing of natural language representations at 4 different

levels of abstraction. A refined version of the vision involves pipelined concurrent processing of the levels with some feedback from higher levels to lower levels.

Since many NLP techniques rely on statistical dependencies in the text, the construction of large-scale, comprehensive data sets, or corpora, has become an important thrust in NLP research. These corpora are composed of a set of texts with words tagged with various labels (e.g. part of speech (POS), semantic, syntactic and role-based ones). Table 2 gives some examples. These corpora provide a rich source for probabilistic modeling of languages. On the other hand, each corpus is limited to a specific domain of a particular language (e.g. English novels and news). The problem of adjusting of either corpora or tools to another domain is yet to be solved, although progress is being made [2-6, 13]. Below we briefly explain each step of our simplified NLP model and provide a description of corpora that are used to derive statistical dependencies.

Table 2. Levels of NLP.

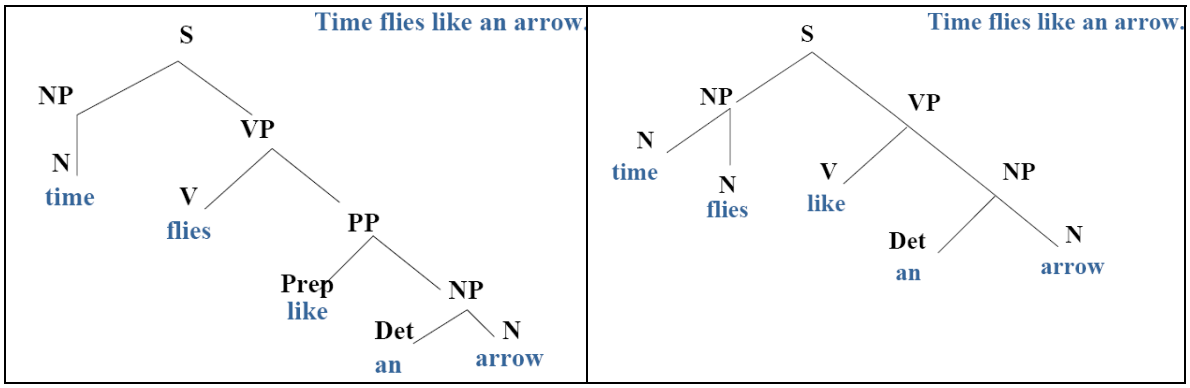
Level	Problems	Methods/KB
Lexical	Part of speech (POS) tagging	Part of speech tagger Corpora: WSJ, Brown Corpus
Syntactic	Generation of parse-trees representing syntactic structure of sentence	Probabilistic parsers; Corpora: WSJ, Brown Corpus
Semantic	Reference resolution; Context modeling; Word-Sense Disambiguation	Semantic parsers, WSD Classifiers; Corpora: FrameNet, SENSEVAL
Pragmatic	Goal, content or topic of a text or discourse; Anaphora Resolution	Text summarization; Text categorization Corpora: WordNet, SummBank

The first step in processing texts is finding word boundaries, called tokenization, and assigning each word a part of speech (e.g. noun, verb, adjective or adverb; quite surprisingly, there are around 40 different POS categories in the most common scheme). This process is called "part of speech tagging" (POS-tagging) and it provides important information for all following stages [14, 15]. Usually, POS-tagging is carried out iteratively using short contextual dependencies that specify how a POS of a given word depends on the POS of the previous word. These dependencies are described by a set of conditional probabilities of the form P(POS1 | POS2) where POS1 is part of speech we are interested in and POS2 is the part of speech of the previous word. Contemporary methods of POS-tagging achieve tagging precision above around 97%.

The second step of NLP analyzes larger chunks of a sentence than individual words. In particular, it identifies Noun Phrases (NP), Verb Phrases (VP), Prepositional Phrases (PP), etc. The corresponding method, called syntactic parsing, outputs syntactic trees that provides both labels and the hierarchical structure of a sentence. Most modern parsers are at least partly statistical; that is, they rely on a corpus of training data which has already been annotated (parsed by hand). In short, they use POS information from a previous level but within a larger context to figure out the conditional probabilities of syntactic

constituents. Parsing methods condition probabilities not just on POSs but also on the words themselves. State of the art in parsing is currently around 92% [6-8]. One of the challenges of syntactic parsing is that each sentence can have multiple valid parse trees. Note, that parsing difficulties can come from propagation of inaccuracies from a previous stage of processing (see table 3).

Table 3. Two alternative parsing trees of the same sentence. In this example, the ambiguity in parsing comes from the wrong assignments of different POS on a previous stage of processing. Even with the correct POS assignment, syntactic tree can vary depending on the attachment of proportional phrases and other factors.



The next step in our simplified NLP model is semantic processing. Issues here concern how to represent the meaning of a sentence, how to make linguistic inferences, as well as word-sense disambiguation (WSD). WSD is the problem of determining in which sense a word is used in a given context [16]. For example, consider the word *bass* that has two distinct senses: a type of fish and a tone of low frequency. In the two following sentences it is clear to a human which senses are used:

1. *The bass part of the song is very moving*
2. *I went fishing for some sea bass*

However, for machines WSD is a difficult task. Compared to POS-tagging, which requires a fairly short context, WSD might involve much longer dependencies. Successful contemporary implementations of WSD use Kernel methods such as SVM trained on the SemCor knowledge base (which contains 352 texts). Most of the texts are annotated with POS, lemma, and WordNet synset. The performance is usually much worse than in POS tagging and is around 75% for English [16, 17]. Such low performance may suggest that contemporary linguistic representations developed for statistical classifiers are not adequate enough to model word senses. One of the solutions is to use better structured input representations that incorporate relations between words such as the ones included in the WordNet [18]. This knowledge base, developed at Princeton University, addresses not only POS and synsets but also such relationships as synonymy/antonymy, meronymy/holonymy (part/whole), hypernymy/hyponym (super and subclasses).

While WordNet describes possible word meanings by corresponding synsets, example sentences, and a rich set of relations there is still a need to automatically identify meaning in a given context. There were several attempts to systematically analyze meaning of words, for example, using argument structure. Levin (1993) proposed that verbs semantic classes correlate with their syntactic and morphological structure. This allowed her to classify verbs in groups such as Put Verbs (mount, place, put) or Correspond Verbs (agree, argue, clash, collaborate, communicate, etc.). However, more detailed examination of Levin's classes revealed that better classification should be at least partially semantically motivated. This started the FrameNet project at Berkeley University. In FrameNet, not only verbs but also other POSs are assigned role frames. The FrameNet lexical database currently contains more than 10,000 lexical units (e.g. "*traffic light*", "*take care of*", "*by the way*"), more than 6,100 of which are fully annotated, in more than 825 semantic frames, exemplified in more than 135,000 annotated sentences.

The basic idea is that one cannot fully understand the meaning of a single word without access to all the essential knowledge that relates to that word. For example, one would not be able to understand the word "sell" without knowing anything about the situation of commercial transfer, which also involves, among other things, a seller, a buyer, goods, money, the relation between the money and the goods, the relations between the seller and the goods and the money, the relation between the buyer and the goods and the money and so on. Thus, a word activates, or evokes, a frame of semantic knowledge relating to the specific concept it refers to, or highlights, in frame semantic terminology (see fig. 1). FrameNet produces much more semantically consistent categories than Levin's classification [19, 20].

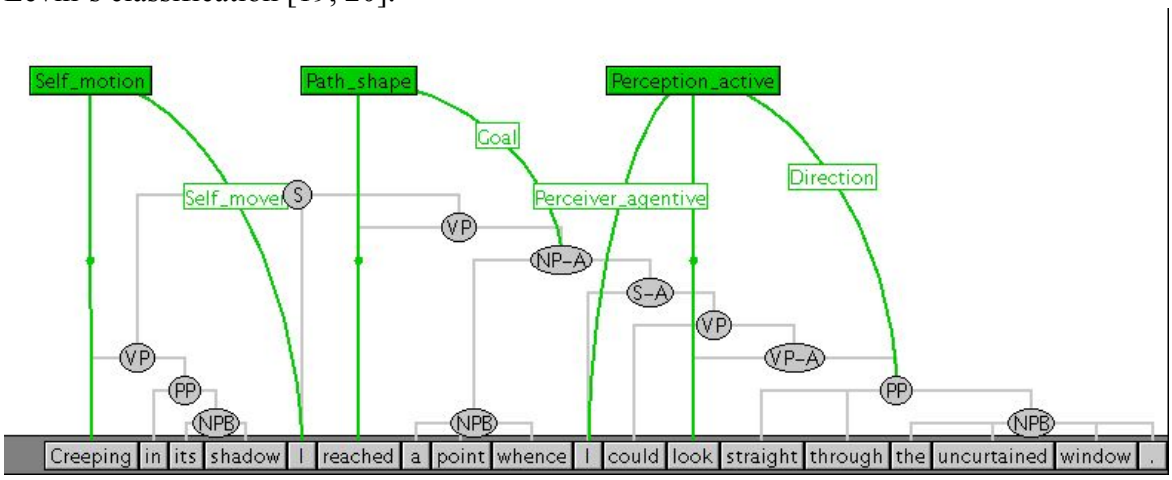


Figure 1. Shallow semantic parser (FrameNet-based by Katrin Erk).

Finally, we turn briefly to pragmatics, which is concerned with understanding the relationships between language and context. For example, an important aspect of this level of analysis is *anaphora resolution*. Simply put, anaphora resolution is concerned with the problem of resolving what a pronoun or a noun phrase refers to. For example, consider the following two cases:

1. John helped Mary. He was kind.
2. There were dresses of several different colors and styles. They were all pretty and labeled with price tags. Sally chose a blue one. Mary chose a skimpy one.

In case 1 "He" clearly refers to John. But to what does "one" refer in case 2? Humans have no problem understanding that the mentions of "one" in the third and forth sentences refer to "they" in the second sentence, which, in turn, refers to "dresses" in the first sentence. However, for a machine, the mentions of "one" could also have referred to "price tags" in the second sentence. For more complex computer communication, like blogs or on-line chat, anaphora resolution is even harder. Other forms of discourse analysis include understanding the discourse structure—i.e., what role does a sentence play in the discourse—and speaker turn-taking.

3. NLP in the Context of RE

NLP in the context of RE differs from general purpose NLP because the inputs and outputs are different, as illustrated in Figure 2. The result of the NLP front end should be a model of the requirements. Although there are a variety of notations and formalisms for requirements, we believe that the structure summarized in this section provides a useful reference model that is close to the mark. For a detailed description and examples see [24]. The requirements are most usefully conceptualized as a database containing structured information, rather than as a text document.

Abstractly the requirements database consists of:
1. *Problem ontology*, which is called an environment model in [24]. This provides an unambiguous vocabulary for defining the requirements: each symbol denotes a unique concept with a well-defined meaning. In our specific framework, a concept can be a type, relationship, attribute, or constant (distinguished instance of a type). Related concepts are generally grouped into modules, often related to types, and are subject to specialization and multiple inheritance that combines constraints by conjunction. Meaning of concepts is described by associated natural language, logical formulas, real-world measurement processes, or links to other defining documents. In particular, concepts can be uniquely mapped into symbols of a typed logic or other formalism to support further analysis, transformation, and simplification.

2. *Requirements hierarchy*. Each node in the hierarchy represents a requirement, which is a constraint that the proposed system will have to satisfy. Nodes can have many views, such as natural language descriptions, diagrams, mathematical formulae, etc. Higher level nodes are more abstract and may leave many details unspecified. Lower level nodes refine the meaning of their parent node by specifying additional details related to the parent requirement. In a completed hierarchy, leaf nodes are defined in terms of the vocabulary of the problem ontology, and are unambiguous in the sense that they do not contain references to undefined concepts. If the requirements are to be used as the basis for automated testing of the system under development, then the concepts used in the requirements must all be measurable or computable from measurable concepts.

3. *System model*. Later stages of requirements engineering generally produce a model of the proposed system at some level of detail. At a minimum, interfaces and externally visible behavior of the proposed system must be modeled, along with its interactions with its context – the (human) stakeholders of the system and external systems it communicates with. There are a variety of notations for this type of model, including use cases, UML, many formal modeling languages, as well as architecture description languages.

The requirements should serve both as guidance for system developers and as a reference standard on which system quality assurance is based. In highly automated processes that current software engineering research is seeking to enable, the information in the requirements should be sufficiently complete and precise to enable automatic generation of at least the software that can test a system implementation to determine whether or not it meets the requirements to within a given statistical confidence level. In some visions of model-based domain specific development, information in the requirements may also be used to directly generate parts of the deliverable code.

In any case, the delivered system is unlikely to be any better than the requirements, reinforcing the mantra that accuracy of the requirements has great importance. Existing manual processes for deriving requirements from informal stakeholder statements therefore incorporate a variety of checking procedures that include reviews, storyboarding, simulation and prototype demonstration, dependency tracing, consistency checking, and many others.

Other processes that must be supported after formalization of the stakeholder input include detecting and resolving conflicts between needs of different stakeholders, finding errors of omission, and finding cases where different stakeholders may agree on the wording of a requirement but not on its meaning. This last case is significant in large systems because they typically involve stakeholders from different specializations.

4. How NLP Can Be Improved in the Context of RE

Generic NLP has only one set of inputs, the natural language text and the accompanying general linguistic resources. In the context of RE there should be additional information: identification of the source of the text, including identity, role in the process, expertise areas, etc. There are also other sources of information, including general purpose built-in information about requirements engineering processes, system development processes, and typical problem domain concepts as well as information about the kind of system to be developed in each particular project. All of this information can be used to limit the search space for the NLP, condition the probabilities of possible word senses, and provide models of the context of the discourse that can provide the basis for judging likelihood of interpretations for much larger bits of text than individual words or phrases. This information can drive post-processing that seeks to identify particular types of errors or just to identify and question the generated interpretations that have weak evidence.

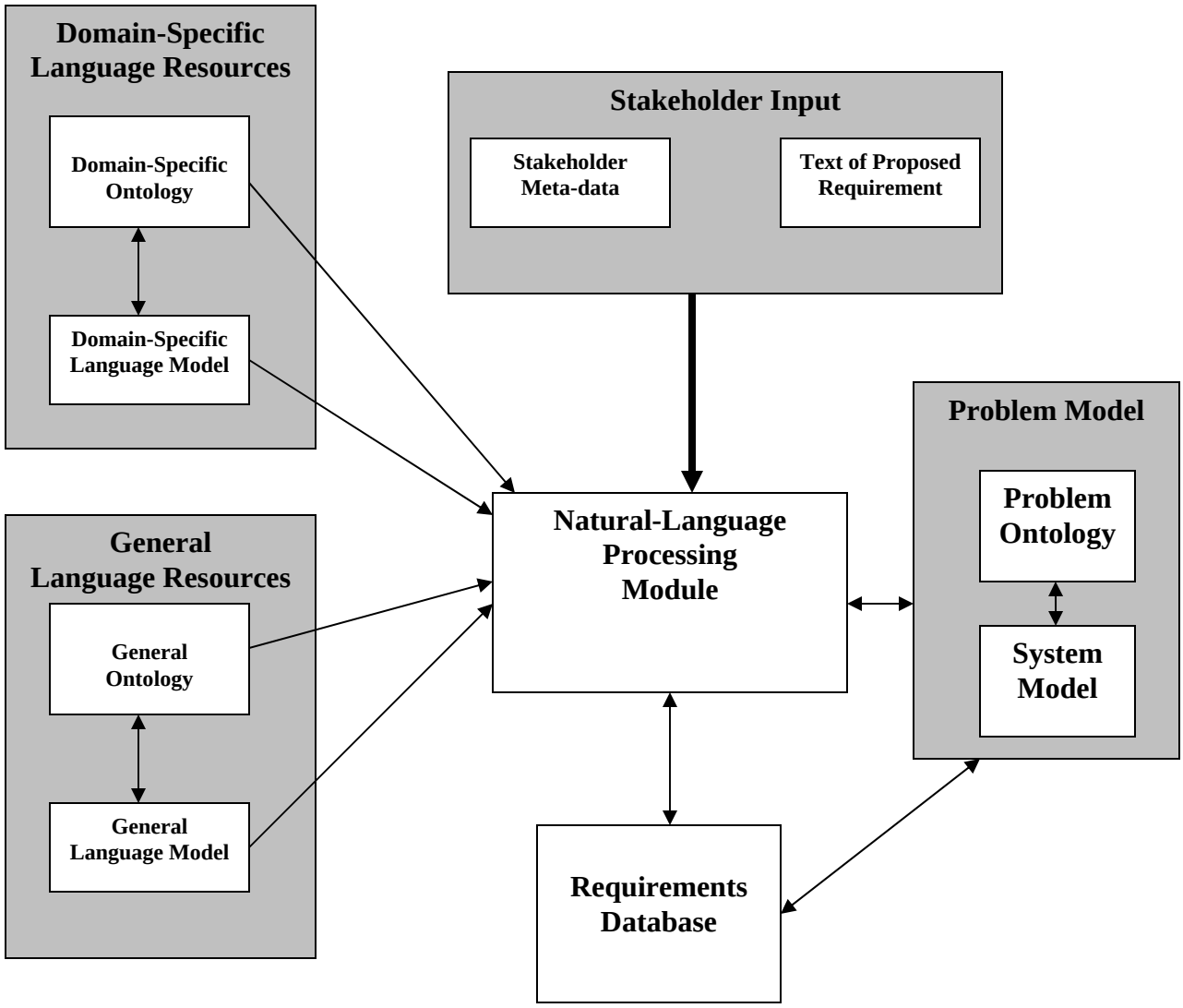


Figure 2. The Connection between NLP and RE processes

5. Conclusions

It appears that NLP is getting close to the point where it can contribute to requirements engineering, but it cannot do so in a vacuum. The results must be checked and reviewed, and existing methods must be improved by using more aspects of the context of the process to improve accuracy.

Even approximate NLP could facilitate text analysis and reduce workload by prioritizing documents, using context for effective search, making summaries, and classifying texts or their fragments even if accuracy of the process is insufficient to support requirements

engineering based on the raw output of the NLP. The difference from fully automated processing is that NLP methods will typically give users several options and it will be their responsibility to select the right one. Thus currently the most safe and effective use of NLP is to integrate its methods with human processing as it is conceptualized in Human System Integration (HSI) framework [21, 22].

The issues that will determine whether or not NLP enters widespread use in requirements engineering are economic: it must cost less and produce more accurate results than corresponding manual processes that rely on human experts to interpret and model the raw statements from the stakeholders. This is a challenging goal that reaches beyond the traditional bounds of NLP to include social, organizational and psychological issues.

References

[1] L.Iwanska, W. Zadrozny (1997). Introduction to the Special Issue on Context in Natural Language. *Processing Computational Intelligence* 13 (3), 301–308.

[2] R. Hwa, M. Osborne, A. Sarkar, and M. Steedman. 2003. Corrected Co-training for Statistical Parsers. *In the Proceedings of the Workshop on the Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining, International Conference of Machine Learning,* Washington D.C.

[3] M. Steedman, R. Hwa, S. Clark, M. Osborne, A. Sarkar, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 2003. Example Selection for Bootstrapping Statistical Parsers. *In the Proceedings of the Joint Conference of Human Language Technologies and the Annual Meeting of the North American Chapter of the ACL*, Edmonton, Canada.

[4] C. Xi and R. Hwa. 2005. A Backoff Model for Bootstrapping Resources for Non-English Languages. *In the Proceedings of HLT/EMNLP-05*, Vancouver, Canada.

[5] R. Hwa. 1999. Supervised Grammar Induction using Training Data with Limited Constituent Information. *In the Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 73-77, College Park, Maryland.

[6] D. McClosky, E. Charniak, and M. Johnson. Effective Self-Training for Parsing. *Proceedings of the Conference on Human Language Technology and North American chapter of the Association for Computational Linguistics* (HLT-NAACL 2006), Brooklyn, New York.

[7] M. Collins. Three generative, lexicalized models for statistical parsing. *In 35th Annual Meeting of the ACL*, 1997.

[8] M. Collins. 2000. Discriminative reranking for natural language parsing. In Machine Learning: *Proceedings of the 17th International Conference* (ICML 2000), pages 175–182, Stanford, California.

[9] M. Marcus, B. Santorini, and M. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[10] J. Kupiec, 1992. Robust Part-of-Speech Tagging Using a Hidden Markov Model. *Computer Speech and Language* 6, pp. 225-242.

[11] B. Merialdo, 1994. Tagging English Text with a Probabilistic Model. *Computational Linguistics* 20(2), pp. 155--171.

[12] G. Niyu, J. Hale, and E. Charniak. 1998. A statistical approach to anaphora resolution. *In Proceedings of the Sixth Workshop on Ven d Large Corpora*, pages 161--170, Montreal (Canada).

[13] M. Steedman, M. Osborne, A. Sarkar, S. Clark, R. Hwa, J. Hockenmaier, P. Ruhlen, S. Baker, and J. Crim. 2003. Bootstrapping Statistical Parsers from Small Datasets. *In the Proceedings of the Tenth Conference of the European Chapter of the ACL*, Budapest, Hungary.

[14] E. Charniak. 1997. "Statistical Techniques for Natural Language Parsing". *AI Magazine* 18(4):33-44.

[15] H. Halteren, J. Zavrel, W. Daelemans. 2001. Improving Accuracy in NLP Through Combination of Machine Learning Systems. *Computational Linguistics*. 27(2): 199-229.

[16] R. Mihalcea, Knowledge Based Methods for Word Sense Disambiguation, *book chapter to appear in Word Sense Disambiguation: Algorithms, Applications, and Trends*, Editors Phil Edmonds and Eneko Agirre, Kluwer, 2006.

[17] R. Mihalcea, Unsupervised Large-Vocabulary Word Sense Disambiguation with Graph-based Algorithms for Sequence Data Labeling. *In Proceedings of the Joint Conference on Human Language Technology / Empirical Methods in Natural Language Processing* (HLT/EMNLP), Vancouver, October, 2005.

[18] G. Miller , R. Beckwith , C. Fellbaum , D. Gross , and K. Miller. Introduction to WordNet: An On-line Lexical Database. *Int J Lexicography* 3: 235-244.

[19] K. Erk and S. Pado: Shalmaneser - a flexible toolbox for semantic role assignment. *Proceedings of LREC* 2006, Genoa, Italy

[20] B. Levin. 1993. English Verb Class and Alternations: A Preliminary Investigation. Chicago: *University of Chicago Press*. Manning, Christopher. 1993.

[21] Plummer S. USAF. (2000). “Memorandum: Awareness of Human-Systems Integration (HSI) in Air Force Acquisitions”.

[22] Blasch, E. Assembling a distributed fused information-based human-computer cognitive decision making tool, Aerospace and Electronic Systems Magazine, IEEE, pp. 11-17, Volume: 15, Issue: 5, May 2000.

[23] Boehm, B. *Software Engineering Economics*, Prentice-Hall, 1981.

[24] V. Berzins and Luqi, *Software Engineering with Abstractions*, Addison-Wesley 1991.

[25] Luqi, L. Zhang, V. Berzins, Y. Qiao, "Documentation Driven Development for Complex Real-Time Systems", *IEEE Transactions on Software Engineering*, Vol. 30, No. 12, December 2004, pp. 936-952.

[26] Luqi, "Transforming Documents to Evolve High-Confidence Systems", *Proceedings of Workshop on Advances in Computer Science and Engineering*, Berkeley, CA, May 6, 2006, pp. 71-72.

[27] Luqi, L. Zhang, "Documentation Driven Evolution of Complex Systems", *Proceedings of Workshop on Advances in Computer Science and Engineering*, May 6, 2006, pp.141-170.

A Learners' Quanta Based Framework for Identification of Requirements and Automated Design of Dynamic Web-based Courseware

Nabendu Chaki[1], Ranjan Dasgupta[2]

[1] Dept. of Computer Science & Engineering, University of Calcutta, Kolkata 700009, India
[2] Dept. of Computer Science & Engineering, National Institute of Technical Teahers' Training & Research, Kolkata 700064, India

{[1]nccomp@caluniv.ac.in, [2]ranjandasgupta@hotmail.com}

Abstract. Learner-centric design of courseware is in the focus of computer aided distance learning for some time now. However, as the courses are more often than not pre-designed, these hardly offer any dynamism and flexibility to cater to the varied need and knowledge background of individual learners. The present work aims towards providing a framework in which the learner would be able to input his learning needs and backgrounds. The framework develops a formal requirements set from this and automates the design of web-based courseware. The collaboration between the stakeholders, i.e., both the learner and the course coordinator, helps improving the quality of the design. In our earlier work, we have proposed a learner-centric, modular approach, named Learners' Quanta (LQ) mode. This LQ model has been utilized and extended here to build the proposed framework. We also developed a prototype for evaluation and further exploration of the idea.

Keywords: Requirement Set, Computer Aided Distance Learning, Learners' Quanta, Knowledge Factors, Reusability, Prototype.

1 Introduction

The identification of specific requirements for individual learners of vastly different background and the design of effective courseware for the use of distance learning are being considered as a challenging problem all over the globe. Several efforts in this regard have been reported which are mostly course-specific.

The non-linear way of storage of information in the form of hypertext has brought a revolutionary change in the teaching-learning process [1, 2, 3, 5, 11]. In the hypertext document, links have been established in such a way that the user can explore, browse and search for not only a particular item but can also get information regarding relevant/associated issues. Cockertion and Shimell evaluated hypermedia document as a learning tool [4]. They focused their study on hypermedia document and included graphical controls for simple interaction behavior. The challenges, from multiple aspects have been well explored by Chen [1]. Many of these works, however,

emphasize more on pedagogical and other issues rather than the identification of requirements.

The working group IEEE "1484 Learning Technology Standards Committee (LTSC)" has designed an architecture called Learning Technology System Architecture (LTSA) to standardize web-based content delivery for all learning technology systems [6]. The LTSA specification follows a basic hypermedia approach with management software working behind to control the sequencing of the hypermedia documents.

Keith S. Taber and his associates [9], at the University of Cambridge put forward a project aimed to integrate English and Science standards using technology as a vehicle. The emphasis, however, was merely to improve the presentation of the learning material. Dr. John Munro [7] of the University of Melbourne has also worked on identification of the requirements for the effective delivery of course content. He essentially tried to analyze learning based on some pre-defined key issues. Guanon Zhang [12] has designed a computer based knowledge system for assisting persons in making decisions and predictions upon human or data-mining knowledge. This work is the closest research to what we are doing. However, in spite of having an almost identical goal to offer maximum flexibility to the individual learner, our approach to meet the target is distinctly different.

The work by S Ray, N Chaki, R Dasgupta [8] presents the LQ model where a course is sub-divided into several topics. The present work is an extension of this LQ model. In this paper, using the LQ model, we have proposed a framework to register the need of the users. A finite set of requirements are identified from this in terms of the target LQs [section 3]. The framework subsequently modifies this initial set of target LQs towards identification of the entire set of LQs that are needed to meet the learning objective. Finally, a proper learning sequence of the selected set of LQs is devised. Besides, the pool of LQs is so designed that it would be able to cater to a wide spectrum of learners with varying requirements. A few screen shots of the prototype that is built and tested following LQ model has been discussed in section 4. Before presenting these details, however, we have briefly reviewed the LQ model [section 2] for the sake of completeness.

2 The Learners' Quanta Model

In this model a course is sub-divided into several topics. A participant may choose one or more topics or the entire course as per his requirement. There shall be a large number of LQs for a subject area or topic of study. These LQs together form a LQ cloud for the specific topic of study. In other words, a topic is constituted by a suitable combination of LQs as chosen by the course coordinator in accordance with the requirement/learning style of the participant.

Learners' Quanta of Study: *A Learners' quantum of study is a measured part of a topic with a well defined set of output objective(s) and requiring a precise input knowledge on the part of the learner*.

Several Learners' Quanta on some topics or sub-topics are being designed and

developed by the authors and they constitute Learners' Quanta Cloud (LQC). In any LQC, there may exist more than one LQs on the same topic or sub-topic written by same author or different authors having same or different output objectives and input knowledge specifications. Based on the requirement of each participant, a subset of LQs are chosen from the LQC and sequencing of the LQs (i.e. LQ chaining) need to be done so that on completion of the entire chain of LQs, the participant can be elevated to the desired level from his initial knowledge level.

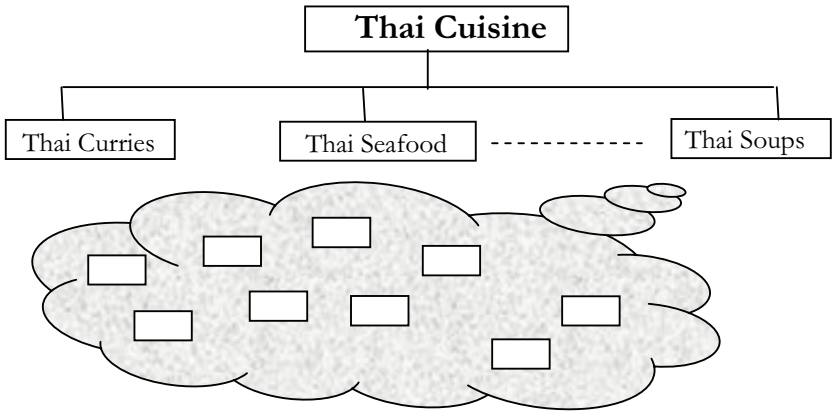


Figure 1: Learners' Quanta and LQ Cloud

2.1 Major Advantages of Learners' Quanta Model:

In this section, we just briefly mention only the major advantages of using the LQ approach keeping in mind the limited space.

Reusability: Design, development of web-based study material is still a costly affair and this can be reduced by reusing the same. In this model, a set of Learners' quanta of study may be reused by different course coordinator for different programs. Naturally more effective course-material may be produced by the author, if he gets his investment back through re-using.

Requirement Identification: The flexibility is increased highly by this approach and the participant can play a significant role in selecting his choice. This in turn leads to evolve a more effective requirements specification.

Cost: The cost of learning may be trimmed off by avoiding un-necessary portion of study. Re-usability also causes a drop in the cost.

Quality Improvement: The collaborative effort of the stakeholders not only improves the effectiveness, but improves the quality of the end-product, i.e., the designed courseware, in this case.

2.2 Terminology used

Before we introduce the algorithm, let us present a formal introduction to all the

terminologies that are referred in the rest of the paper.

Learner's Quanta (LQ): A Learner's Quantum of study is a measured part of a topic with a specific output objective requiring a specific input knowledge on part of the learner.

Learner's Quanta Cloud (LQC): A Learners' Quanta Cloud (LQC) is a collection of semantically related group of learners' quanta. Any arbitrary quanta, LQi could be part of more than one LQC, where LQs are grouped based on different semantics. In simpler terms, LQC is a set of related LQs.

As illustrated in figure 1, this semantic relation could be, e.g., that two LQs are both related to learning on Thai seafood cooking.

Knowledge Factor (KF): A Knowledge Factor (KF) is an atomic element of information. Each LQ is associated with a unique set of input KF and another set of output KFs. The intersection of the input and output set for a particular KF is usually a Null set.

Target Knowledge Factors: The Target Knowledge Factors (TKF) are the set of KFs specified by the user as the set of output objectives he/she wishes to acquire.

Known Knowledge Factors: The Known Knowledge Factors (KKF) are the set of KFs specified by the user as the set of already known elements of information for a particular learner.

LQ Dictionary: The LQ dictionary for a subject area refers to the entire set of LQs with their corresponding KFs stored in one specific place. The dictionary is different for each subject area.

3 LQ based Requirements Specification and Designing

Given

I: Input knowledge set of the learner
I is the set of KKFs as defined in section 2.2.

R: Requirement knowledge set specified by the learner
R is the set of TKFs as defined in section 2.2.

Oi: Objective set to be attained by a learner on completion of the ith LQ

Pi: Pre-requisite knowledge set for the ith LQ

In a real life situation, the intersection of R and I would produce a Null set. Now the problem is to identify the required set S of minimal LQs from the available LQ cloud so that any learner with given Input knowledge background I can reach the level R. The condition of minimalism depends upon various factors according to the requirements and preferences of the learner. The metric for minimalism could be duration of learning time, total cost of learning or just the number of LQs in the proposed solution.

Whatever be the condition of minimalism, in order to identify the required set S of minimal LQs, the system has to find an initial set S_1 of k_1 LQs from the LQC, such that the set union of the objectives of the derived set S_1 of LQs minimally covers R, i.e.

$$\bar{U}Oi \supseteq R, (1 \leq i \leq k_1) \quad (1)$$

There may be zero or more sets of LQs for which the condition 1 mentioned above are true. In the event that no such set of LQs exists which meets condition 1, the LQC under consideration cannot provide any solution for the proposed learner requirement.

After the k_1 number of LQs for the initial solution set is made available, we do a backward search amongst these k_1 LQs in the LQ cloud to finally reach the Input knowledge set (I) of the learner. However, in this process, if we do not find any such single LQ or multiple LQs, we can conclude that the LQs in set S_1 does not provide any sequence of LQs from the existing LQ cloud to form a path for the learner with Input knowledge level I to reach the specified requirement level R.

Once the k_1 number of LQs for the initial solution set is made available, we have to look at a sum of individual pre-requisite knowledge for each of these k1 LQs. Let Q_1' represents this, i.e.

$Q_1' = \bar{U}Pi$, $(1 \leq i \leq k1)$ (2)

If the learner's input knowledge set (I) of the learner covers Q_1', then our task is almost over, i.e., we could identify the set of LQs that this learner requires. In other words, the initial set of LQs S_1 has been identified as the final and target set S at the end of just a single iteration. However, in a real life situation, more than often we would find that the learner's input knowledge set I cannot cover set Q_1'.

In order to find the complete list of LQs from the initial list, backward searching through each of the LQs satisfying 1 is required. This is done as follows:

Considering I as the Input knowledge set of the learner, in order to learn the k_1 selected LQs, the revised set of requirements will be,

$Q_1 = Q_1' - I$ (3)

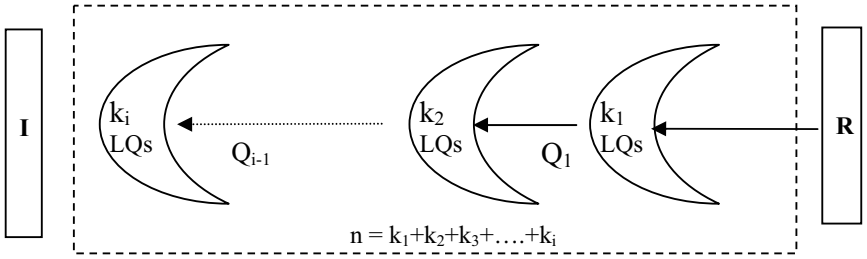


Figure 2: Identifying the required minimal set of LQs

This set of pre-requisites Q_1 is beyond the input knowledge of the proposed learner and is available from the LQ cloud under consideration. Thus the system would identify a new set of k_2 LQs that minimally covers Q_1. The learner, therefore, has to study these k_2 additional LQs to build the pre-requisite knowledge level as required to study the set S_1.

These new set S_2 of (k_1+k_2) LQs in total is thus identified in this second iteration as the revised set of minimal LQs that are to be studied. As the pre-requisite knowledge for the first set of k_1 LQs are to be covered by the sum of the Objective sets of k_2 additional LQs and I together, the problem reduces down to finding the pre-requisite knowledge for these k_2 LQs (see figure 2) and to cross-check if that is covered by the learner's input knowledge I.

At this stage of the processing, the system would take a set union over the individual pre-requisite knowledge for each of these additional k_2 LQs to obtain Q_2'. The iterative process is continued till a set of k_g additional LQs is identified in the g^{th} iteration whose set of pre-requisites, Q_g are covered entirely by I. The set S_g derived up to this stage, is to be marked as the solution set S. If the cardinality of this final solution set S, is n then, $n = (k_1 + k_2 + k_3 + \ldots\ldots + k_g)$.

Once the LQs are identified, the sequence through which the learner will proceed must be tracked. For this, we generate pre-requisite directed graph with LQs as node and the edges are formed by the pre-requisite data. Thus if P & Q are two LQs where P is the pre-requisite LQ for Q, we draw an edge from P to Q. If we draw the graph of the LQs as explained above, we may reach a number of disjoint acyclic digraph with multiple zero-pre-requisite LQs and with multiple finish-LQs.

By the term zero-pre-requisite LQ, we identify those LQs, for which the participant has the Input knowledge, i.e $Pi \subseteq I$ and the finish-LQ represents those LQs for which intersection of R and Oi is non-null and not a pre-requisite of any other LQ in the graph. However, there may be multiple nodes in the graph for which intersection of R and Oi is non-null.

Finally, to derive the sequencing of the LQs we use the topological sort by identifying successively nodes with zero in-degree and removing the edges drawn from it before the second iteration. However, if we reach nodes with equal sequence number, we can offer those LQs in parallel.

4 Developing the Prototype

In order to validate the proposed design, we have developed a prototype for the system. We have used an object-oriented platform for the purpose. The prototype has been developed in three stages. At first, the class modules have been compiled into a Dynamic Link Library (DLL).

In the next phase, a standalone ActiveX component LQBox is defined. A setup package has also been made for this control such that any user can use this component as a black-box without worrying much about the implementation logic behind the scenes. The LQBox component is also compatible with all the MS languages.

Finally, an application is developed to depict the working of the modules mentioned above. Data entry, validation and solution generation can be hence shown. Using this library as a reference makes all the classes available.

4.1 Architecture of the Prototype

We have implemented the system in a three tier, modular, layered framework. It conforms to the LTSA guidelines [6] and performs the job of a “system coach”. The architecture has the following characteristics.

Task directed: The design of the entire system has been such so as to assist in the systematic approach to solving the adaptive course scheduling problem

Modular: The entire system is modularized such that each module does one

specific task. Good cohesion between modules has been maintained.

Layered: The system has been developed as a three tier application. The bottommost layer is the backend layer. The LQ Dictionary as well as the Target and Known Knowledge Factors may be kept in a database or in a text file. The main focus of the work is to develop the middle layer. It has been implemented as a DLL and a component is associated with it. The top layer is basically a "wrapper layer" providing the necessary user interface for use.

Component based: The use of a component based approach helps create modular and reusable system. One can build upon an existing component such that productivity can be vastly increased.

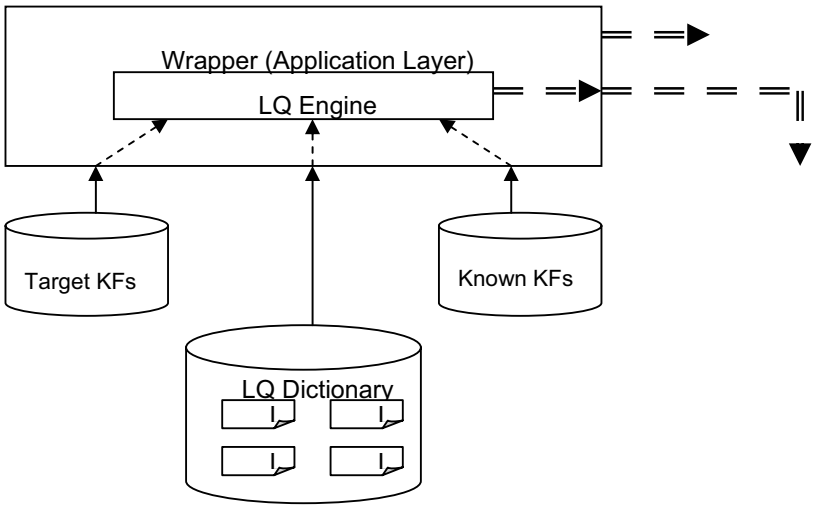


Figure 3: Architecture of the system

Extendible: As the system conforms to the LTSA architecture, is modular and reusable in nature, it may be extended by adding further layers. Expandability can be both horizontal and vertical. More components may be added both in conjunction with the existing or as wrappers to the system.

User friendly: Each of the modules clearly specifies their input and output requirements. Error reporting is also managed using an error model. The UI has been intuitively designed such that ease of usage of the system is greatly enhanced.

The development of the logic of the system was thought of in a step-by-step manner. First of all, the classes containing data were defined. This included the design of class LQ, class LQDictionary, class KFSet. and the class LQEngine. All of these classes have been compiled into a Dynamic Link Library (DLL).

The ActiveX component is the actual "Black Box" of the prototype that controls the functioning. We have used an MS FlexiGrid Control to display the LQs generated in every pass. We call this component the LQBox. A setup package has also been made for this control such that any user can use this component as a black-box without worrying much about the implementation logic behind the scenes. The

LQBox component is also compatible with all the MS languages.

The task of searching for a minimal LQ set is done by the LQDictionary. It returns a new LQDictionary object which is a subset of the Original LQDictionary. The job of successively refining the Target KF set, checking with the Known Kfset and with the basic LQs is done by the LQBox. On the whole, the system has been modeled as a set of components that can be reused. If one needs just the class descriptions, a reference to the Class Description file will suffice. If one needs the component for visual display and the complete course path generation, an inclusion of the LQBox component will suffice. The LQ Engine window has the LQBox embedded in it.

5 Conclusion

This paper attempts to identify the requirements and automate the design of courseware based on the need of individual learner. Based on the input knowledge level of the participant as well as the desired output, the LQ requirements are identified using an iterative methodology. The proposed framework further derives the design of the courseware by detecting the learning sequence for the selected LQs. This naturally will make this adaptive system more acceptable to the participants.

The focus of developing the prototype is to lay a foundation for further work in future. The heart of the system is complete and the engine satisfies the requirements of reusability. The prototype offers one of the possible minimal solutions that exist. While the learning plan generated is not necessarily an optimal solution from the perspective of cost or time for learning, it surely is irreducible and no redundancy exists. The solution hence satisfies our primary objective.

References

1. Chen, B. : Challenges in Experimental Research Conducted in a Fully Web-based Distance Learning Environment : Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (2006), pp. 1778-1783.
2. Mayer, R. : The promise of multimedia learning: Using the same instructional design methods across different media. Learning and instruction, Vol. 13, (2003), pp. 125-139.
3. Kurbalija, J.; Dincic, D.; Slavik, H. : Interactive hypertext in e-learning: a case study asynchronous learning in the online classroom : Proceedings of the 5th International Conference on Information Technology Based Higher Education and Training, (2004), pp 15-20.
4. Cockerton T., Shimell R. : Evaluation of a hypermedia document as a learning tool : Journal of Computer Assisted Learning, Vol. 13, No. 2 (1997).
5. J. R. Laleuf, A. M. Spalter : A Component Repository for Learning Objects: A Progress Report : First ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'01), pp. 33-40.
6. IEEE LTSC : Learning Technology Systems Architecture (LTSA), Draft 9,. URL: http://www.edutool.com/ltsa (2001).
7. Munro John : Facilitating Effective Learning and Teaching : Proc. of Technology Colleges Trust Online Conference (2002).
URL:www.cybertext.net.au/tct2002/keynote/printable/munro.htm

8. Ray S., Chaki N., Dasgupta R. : Design of an adaptive web-based courseware : Proc. of IASTED International Conference on Intelligent Systems & Control (ISC 2004), Honolulu, Hawaii, USA, (2004).
9. Taber K. S. : Development of Student Understanding: A Case Study of Stability and Lability in Cognitive Structure, Research in Science & Technological Education, Vol 13 (1), (1995) pp.87-97.
10. Vassileva J., Deters R. : Dynamic Courseware Generation on the WWW : British Journal of Educational Technology, Vol. 29, No.1 (1998).
11. Ehsan Sheybani, Gita Javidi. : Interactive Multimedia And Distance Learning : Proc. of the 34th ASEE/IEEE Frontiers in Education Conference : USA. (2004), pp. SID-1 - SID-3.
12. Zhang George Guanon : Computer Based Knowledge System in the USPTO United States Patent Office, September, 2004. URL: http://www.uspto.gov/Patents/United States Patent 6,795,815.htm

Transparency, Simplicity, and Trusted Software

Daniel E. Cooke, J. Nelson Rushton, and Brad Nemanich

Department of Computer Science
Texas Tech University

Abstract. In addition to generating correct codes from specifications, trusted system specifications need to be transparent.

Introduction.

Many Computer Scientists seem to have adopted the view that complicated tools are needed to solve complicated problems. Although there are advantages to the current problem-solving environments there remains a need to develop simpler tools, especially simpler languages. The SequenceL language has been developed with the goal of eliminating the need to focus on unnecessary technical details and complexities that distract one from the problem to be solved. [C96, CR] It is becoming increasingly clear that the complex tools we use today will not meet future demands of many critical applications.

Two quotes seem to sum up opposing views on computer languages. According to S.P. Jones, one of the developers of Haskell, "Any language large enough to be useful is dauntingly complex." [Jones] On the other side we have C.A.R. Hoare's quote from his Turing Award lecture, 27 years ago, "The price of reliability is the pursuit of the utmost simplicity." [Hoare] It is interesting that Hoare's view was probably the predominating view when he stated it. In the intervening years we have seen the growth in Object-Oriented Programming and dauntingly complicated desktop systems, leading to Jones' view at the turn of the 21st century. We posit that Jones' view is widely held today. Much of the Computer Science community is vested in research to help organize the complexity of the dauntingly complex languages we use to organize the complexity of problem solving.

We claim the troubles with existing tools are twofold. First is the gap or distance between the mental model of a user and the requirements, and second is the distance or gap between the requirements and the executable code. Currently, the gulf between requirements and code is responsible for between 35% and 75% of software flaws, tending toward the higher number: [Jones2000], [Grady87]. However, in the long run the gulf between one's intuition and requirements may be more important, since this is the step that can't be certified by explicit deduction.

The work we have focused on over the past 20 years remains inspired by Hoare's view. We believe that the increasingly complex nature of current and presumably future tools stand in opposition to the solution of complicated problems, and that only through the development and use of exceedingly simple tools will we be able to solve, rapidly and dependably, the complicated problems we will see in the future. This paper gives examples that make our view plausible, by showing where

SequenceL enables substantial simplifications over currently used techniques. It should be noted that these examples are extracted from SequenceL as it is being applied to prototype critical Guidance, Navigation, and Control Systems at NASA Johnson Space Center.

If we accept Hoare's claim – that simplicity is the key to the development of trusted software – then what is the key to simplicity? We believe that languages must be small and concise. SequenceL, the language we have developed, can be reduced to roughly 9 grammar rules. You can carry the language around in your head – you don't need reference materials that impede problem solution. You don't need to remember how operators specific to the language work.

Equally important is that the semantics of the language be very simple. Apart from implementing operators that have intuitive meanings, SequenceL possesses two simple computational laws (the Consume-Simplify-Produce and the Normalize-Transpose). All structures in the languages are producible simply from these semantics. [CR] Not only does this simplify the development of assurances concerning the synthesis of codes, there are *no* operators that require special definitions (like the *maps* and *zips* of Haskell). [HJW92] All the operators are intuitive. This leads to a major requirement for trusted software development – that problem solutions are transparent – that a problem solution can be seen, in a glance, to do what you want.

Motivation

In the past, human-rated NASA missions have been heavily scripted, meaning that hardware and software systems are unlikely to be capable of dealing with many unanticipated events. This condition further means that the primary responsibility for handling unforeseen situations resides with humans, who are either onboard the spacecraft or in mission control.

Missions currently being planned by NASA's Exploration Systems Mission Directorate will not be pre-scripted. These missions will be less predictable than past missions and some will be of much longer duration and involve considerably greater distances. Minute-by-minute communication between Mars and Mission Control will be physically impossible given roundtrip communication delays ranging from six to forty-one minutes, not including a solar conjunction involving Mars and Earth during which communication may be impossible for up to three weeks.

Hardware developed for future missions is likely to be of greater capability to take advantage of the increased functionality software can serve. Thus, systems will be more adaptable for varied mission scenarios. An ability to rapidly and dependably develop and modify software could provide NASA the means to alter system capabilities on the fly. Following current practices, software modifications to space-based systems can take in the months or years to make. *To modify capabilities between and during missions, revolutionary software development approaches are needed.* [CBLG06] New approaches that, in the tens of minutes, can result in effective and dependable modifications are needed. The SequenceL research is focused directly on these challenges.

Immediately Trusted Systems

NASA will require *immediately trustable systems:* system components that can be specified and generated in a matter of minutes. To meet these demands of future NASA missions, there are two gulfs that must be bridged. The first gulf is the long studied gulf between specifications and the programs intended to satisfy them. We are making novel strides in bridging this gulf through auto-coding. Clearly, auto-coding is not a new idea, but our language tends to make certifications about the auto-coder easier. First, SequenceL itself is a very small language, consisting of roughly 9 BNF rules. Secondly, the two computational laws that comprise the semantics of SequenceL are very simple. We have recently shown that once these two simple laws are defined, all remaining definitions can be implemented simply in terms of them. The laws are the *Consume-Simplify-Produce* and the *Normalize-Transpose* (CSP-NT) and are presented in detail in other papers. [CR] Theoretically, CSP by itself would be sufficient for Turing Completeness; but the addition of NT simplifies the encodings of the language itself and the codes written in it, by allowing us to frequently avoid the use of the recursion sledgehammer to swat the iteration fly. Recursion is generally more complex, semantically and intuitively, than its special cases such as iteration and NT [Dijkstra1990].

The second gulf that must be bridged to meet the demands of immediately trusted systems is **transparency,** which means that one can determine what is specified, at a glance or with minimal effort. Bridging the gulf between the specification and the code generated, addresses *verification*. Bridging the gulf between the specification and the user's intentions, addresses *validation*. Transparency is the key to this concern since it addresses the amount of effort that goes into ascertaining what a specification means. Consider the following examples of specifying Jacobi Iteration. The actual specification for one iteration is:

$$\mu_{j,k}' = (\mu_{j+1,k} + \mu_{j-1,k} + \mu_{j,k+1} + \mu_{j,k-1})/4 - ((\rho_{j,k} * \Delta^2)/4)$$

when neither j or k subscript the first or last row or the first or last column

where μ and ρ are input matrices for each iteration and μ' serves as μ in the next iteration. Haskell code appears below for the algorithm:

```
jacobi a delta =
   map (\i -> [jacobi_helper a delta i j| j<-[0..(length (a!!i))-1]], [0..(length a)-1])

jacobi_helper a delta i j
   | i==0 || j==0 || (length a)-1 == i || (length (a!!i))-1 == j = a!!i!!j
   | otherwise =
         (a!!(i+1)!!j + a!!(i-1)!!j + a!!i!!(j+1) + a!!i!!(j-1))/4 - ((a!!i!!j) * delta^2)/4
```

We claim that most of the effort necessary to see that this code is correct goes into analysis, not of whether the equation is transcribed correctly, but of the second order constructs used to traverse the data structure and apply the equation: map, lambda abstraction, list comprehension, and ellipses, nested four levels deep.

Normally specified equations like the one above are nicely – even transparently – stated in **Matlab**. Unfortunately, when it comes to Jacobi, the

constraints on the subscripts require one to dip into the procedural language accompanying Matlab:

```
function res = ji(a, b, delta)
for i=1:length(a)
   for j=1:length(a(1,:))
      if (i==1 | i==length(a) | j==1 | j==length(a(1,:)))
         res(i,j) = a(i,j);
      else
         res(i,j) = (a(i-1,j) + a(i+1,j) + a(i,j-1) + a(i,j+1))/4;
      end
      res(i,j) = res(i,j) - (b(i,j)*delta^2/4);
   end
```

Having a predefined, reusable function or object has drawbacks as well since it is one of thousands, which both the author and reader of the code must remember. Furthermore, there may be a particular approach the designer wishes to deploy for the computation. Finally, however vast the library, novel problems will arise which require use of the core language features.

In SequenceL one can see that the solution maps directly to the specifying equation with minimal effort:

```
jacobi J,K(matrix a,b, scalar delta) :=
      (a(J+1,K) + a(J-1,K) + a(J,K+1) + a(J,K -1)) / 4 - ((b * delta^2)/4)
            when not((J =1 or size(a)=J) or  (K=1or size(a)=K)) else a(J,K)
```

Notice all of the operators are intuitive and no specialized functions need to be known. Consequently, the SequenceL function aligns well to the specifying equation. Furthermore, a natural language definition tends to spring from the SequenceL definition: *The (j,k)th element of the resulting matrix is the average of its surrounding elements below, above, to the right, and to the left, when j is not the first or last row and k is not the first or last column.*

Principles of SequenceL

One of the ways we are currently trying to improve upon the transparency of SequenceL is by providing *principles* one can follow in translating formulae into SequenceL specifications *without* having to know anything about the underlying computational laws. The whole point of transparency is this: can you trust a specification – even if it generates correct code – if you have to do lots of work to see that the specification states what you want computed?

To achieve better transparency there is a need to address information hiding – similar to the principles David Parnas developed decades ago. Essentially, Parnas' principles concern compartmentalization – that the user of an operation have no knowledge of its implementation *and* that the implementer of an operation have no knowledge of particular applications. [Parnas] In SequenceL, the CSP-NT are abstract semantics that enable all of the non-primitive SequenceL constructs. There

are no specialized operators requiring specialized knowledge about the operator's semantics. The only semantics the SequenceL user would need to know are the CSP-NT. By developing principles for using SequenceL, our goal is to provide a shield for the user – so that he/she need not know semantics at all.

Initial efforts have focused on equation-based specifications like the Jacobi example above. We have explored equations beyond numerical computations to include ones from set-builder notation, denotational semantic equations, etc. In all cases explored so far, one can simply translate the equation directly into SequenceL. Examples from set builder notation follow:

in(**scalar** X, **vector** S) := **or**(X = S(I))
{*Or results in an existential operator based on I*}

subset(**vectors** S1,S2) := **and**(in(S1(I),S2))
{*And results in an existential operator based on I* }

intersection(**vectors** S1,S2) := {X **when** in(X,S1) **and** in(X,S2)}

The Jacobi iteration example, given above, uses *free variables* J and K to implement a solution. However, in certain cases the encodings may be further simplified. An n-ary function *f* acts *pointwise* on its *i*'th parameter *x* if every occurrence of x in the definition of f either has no free variable subscripts, or is subscripted by free variables identical to those subscripting *f* on the left hand side. If a function acts pointwise on all its inputs, free variables may simply be erased from both sides of the equation defining a function, and SequenceL's NT semantic will restore them automatically. For example, matrix addition may be defined in SequenceL as

matplus$_{I,J}$(**matrix** A, **matrix** B) := A(I,J) + B(I,J)

or, equivalently through NT, as

matplus(**matrix** A, **matrix** B) := A + B

Another principle for using SequenceL involves a determination of what varies and what does not vary in a specification. This principle often results in summarizing equations. Consider the following example to compute Euler's number:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

Notice that the numerator remains constant and the denominator varies. In SequenceL, the n^{th} approximation of e is given as:

E(scalar N) := sum(1/fact([0,...,N]))

Note, that translators for SequenceL execute these specifications and some even generate code. Finally, we have found that in the majority of cases, iteration and recursion is used to break apart and put together nonscalar structures. In general, when using SequenceL this assembly/disassembly can be ignored and one can focus strictly on the basic computation.

Natural Language.

We have only recently initiated studies of the relationship between Natural Language specifications and SequenceL. We are focused on requirements for abort systems we are developing in collaboration with NASA Johnson Space Center's Guidance, Navigation, and Control engineers. We recently completed the Shuttle Abort Flight Management System in SequenceL. The original system prototype took six months of full-time effort for one GN&C engineer. Once the system was validated, the team turned over the requirements document to General Dynamics to develop the flight-certified code, which took roughly two years to complete and cost approximately $8 million.

Working mainly with the requirements documents, and with minimal contact with the JSC GN&C engineers, SequenceL specifiers developed an understanding of the requirements and elaborated, executed, and validated them in SequenceL with six weeks of one person's full-time effort. The GN&C engineers agreed that, although not identical, our efforts and theirs were comparable in difficulty and scope. Thus, in our initial experiment, we completed the requirements review and produced a working prototype in SequenceL in one-fourth the time it took the GN&C engineers for the original SAFM development.

The function *Sort TAL sites* is one of the most complicated SAFM components. It arranges the top four sites according to pre-defined priorities. TAL stands for Transatlantic Abort Landing. These are targeted landing options available in the event of an abort during the ascent phase. What follows are the executable requirements in SequenceL. After each component of the requirement, we have reversed engineered a Natural Language (NL) requirement:

SORT_TAL – component 1

```
preferred(vector B, A, scalar Prime) :=
                B when and([B:subscript = 4,B:available,B:valid]) else
                [ ] when and([A:subscript = 4,A:available,A:valid]) else
                B when and([B:location = Prime,B:available,B:valid]) else
                [ ] when and([A:location = Prime,A:available,A:valid]) else
                B when and([B:throttle < A:throttle,B:available,B:valid]) else
                B when and([B:throttle < A:throttle,B:available]) else
                B when A:subscript > B:subscript
```

Natural Language:

1. Site B is preferred to site A if B is the 4th site and is available and is valid, however Site A overrides B if it is the 4th site and is available and is valid; otherwise

2. B is preferred to A if B is the prime site and is available and is valid, however Site A overrides B if it is the prime site and is available and is valid; otherwise
3. B is preferred if B's throttle value is less than A's throttle value and B is available and is valid; otherwise
4. B is preferred if B's throttle value is less than A's throttle value and B is available; otherwise
5. B is preferred if A was entered after B in the list of sites

Only after the Natural Language specification was produced from the SequenceL did it become apparent that conditions 3 and 4 above, can be collapsed with the weaker condition (4).

SORT_TAL – component 2

```
sort_tals( matrix [Site|Sites], scalar Prime) :=
        Site  when size([Site|Sites]) = 1  else
          appends(sort_tals(preferred(Sites,Site,Prime),Prime), [Site],
                    sort_tals(complement([Sites,preferred(Sites,Site,Prime)]),Prime))
```

Natural Language: Sort a set of sites choosing the first site *S* and returning it if it is the only site, otherwise append the following results:

1. The sorted list of all those sites preferred compared to *S*;
2. Site *S*; and
3. The sorted list of all the sites not preferred when compared to *S*.

SORT_TAL – component 3

$geoloc_{I,J}$(**matrix** Sites) **:=** Sites(I) **when** Sites(I):loc = Sites(J):loc **and** I <> J

Natural Language: Return any I^{th} **site** when there is a J^{th} **site** such that I≠J and the location of **site** I is the same as the location of **site** J.

SORT_TAL – component 4

```
stals(matrix Sites, scalar Prime)  :=
      sort_tals(Sites,Prime) when geoloc(Sites) = empty else
          complement(sort_tals(Sites,Prime), geoloc(sort_tals(Sites,Prime))(2)) ++
          [geoloc(sort_tals(Sites,Prime))(2)]
```

Natural Language: Sort all TAL sites when there are no co-located sites **otherwise** add the least preferred co-located site to the end of the sorted list of all the other sites

The requirements for the *Sort TAL sites* function were originally stated by NASA engineers in natural language, quite different from that derived from the SequenceL, and conforming more to a procedural paradigm. We conjecture this was because they were thinking already in terms of the implementation language, in this case *C*. We are engaged now in the exercise of extracting natural language specifications from the SequenceL to identify patterns and a restricted grammar for producing SequenceL from Natural Language Specifications and vice versa.

Summary and Conclusions

The goal of the SequenceL effort is to achieve transparency in specification and truth preserving synthesis of procedural codes from those specifications. [1] Key to the synthesis concern is the fact that SequenceL's semantics are simple and based on only two computational laws.

Key to transparency is the development of a small set of principles to guide the designer when working with formal and natural languages. Systems that are more immediately trustworthy must produce correct codes from transparent specifications. In previous papers SequenceL has been compared to a number of languages including *FP* [B78], *Gamma* [BL93], and *NESL* [B96]. SequenceL has been used to prototype significant NASA GN&C Systems including SAFM, the ORION Onboard Abort Executive, and the Shuttle's Ascent Predictor Software.

References

[B78] John W. Backus: Can Programming Be Liberated From the von Neumann Style? A Functional Style and its Algebra of Programs. Commun. ACM 21(8): 613-641 (1978)

[BL93] Jean-Pierre Banatre and Daniel Le Metayer, "Programming by Multiset Transformation, January, 1993, Vol. 36, No. 1. *Communications of the ACM,* pp. 98-111.

[B96] Guy Blelloch, "Programming Parallel Algorithms," March, 1996, Vol. 39, No. 3. *Communications of the ACM,* pp. 98-111.

[C96] Daniel E. Cooke, "An Introduction to SEQUENCEL: A Language to Experiment with Nonscalar Constructs," *Software Practice and Experience,* Vol. 26 (11) (November, 1996), pp. 1205-1246.

[CBLG06] Daniel E. Cooke, Matt Barry, Michael Lowry. and Cordell Green "NASA's Exploration Agenda and Capability Engineering," COMPUTER (January 2006) Vol. 39 No. 1, pp. 63-73.

[CR] Daniel E. Cooke and J. Nelson Rushton, "Normalize, Transpose, and Distribute: A Basis for the Decomposition and Parallel Evaluation of Nonscalars," in press *ACM Transaction on Programming Languages and Systems.*

[Dijkstra 1990] Dikjstra, Edsger and Scholten, Carel. *Predicate calculus and program semantics.* Springer-Verlag New York, Inc. 1990.

[Grady87] Grady, Robert B. 1987. "Measuring and Managing Software Maintenance.: IEEE Software 4, no. 9 (September): 34-45.

[Hoare] C.A.R. Hoare, ``The Emperor's Old Clothes.'' *Communications of the ACM*, Vol. 24, No. 2, February 1981, pp. 75-83.

[HJW92] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, et. al.: Report on the Programming Language Haskell, A Non-strict, Purely Functional Language. SIGPLAN Notices 27(5): R1-R164 (1992)

[Jones] "Wearing the hair shirt A retrospective on Haskell," research.microsoft.com/ ~**simon**pj/papers/ haskell-retrospective/HaskellRetrospective.pdf.

[Jones2000] Jones. Capers. 2000. *Software Assessments, Benchmarks, and Best Practices.* Reading, MA: Addison-Wesley.

[Parnas] Parnas, D.L. "On the Criteria To Be Used in Decomposing Systems Into Modules" *Communications of the ACM*, Vol. 15, No. 12, pp. 1053-1058, December, 1972.

Support from NASA: NNG05GP48G, NNG06GJ14G, and NNG06GJ14G.

[1] To support, respectively, validation and verification.

Towards Combining Ontologies and Model Weaving for the Evolution of Requirements Models

Allyson M. Hoss and Doris L. Carver
Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803, USA
ahoss1@lsu.edu, dcarver@lsu.edu

Abstract. Software change resulting from new requirements, environmental modifications, and error detection creates numerous challenges for the maintenance of software products. While many software evolution strategies focus on code-to-modeling language analysis, few address software evolution at higher abstraction levels. Most lack the flexibility to incorporate multiple modeling languages. Not many consider the integration and reuse of domain knowledge with design knowledge. We address these challenges by combining ontologies and model weaving to assist in software evolution of abstract artifacts.

Keywords: abstract artifact, design knowledge, domain knowledge, modeling languages, knowledge reuse, software evolution.

1 Introduction

Generally speaking, software evolution refers to changes a software product undergoes to meet changes in its environment and/or requirements. Modifications include adding, deleting, and/or modifying artifacts such as requirements, design, source code, and test cases. We address three challenges in software evolution that receive little attention from current research: evolving high-level artifacts such as requirements and design models; reusing software design and domain knowledge; and, integrating multiple software modeling languages during software evolution.

Software evolution techniques include ad-hoc copy-and-modify, refactoring, visualization, generative, and aspect-oriented approaches. Most of these approaches address primarily source code changes. One of the current challenges in software evolution is the need to incorporate techniques to evolve the more abstract artifacts of software development [1]. Additionally, the few software evolution approaches that do abstract above the code level are often "far too detailed to be helpful for talking about the design with someone else…working with too detailed a model is a trap" [2].

A second limitation common in software evolution techniques today is their single language focus. Few consider the plethora of modeling languages ranging from general-purpose languages to domain specific languages. It is not uncommon for software applications to incorporate several modeling languages, especially as a result

of evolving technologies. A "crucial, and largely neglected, aspect of software evolution research is the need to deal with multiple languages" [1].

Lastly, there is a lack of software design and domain knowledge reuse in most software evolution approaches. A recent description of model evolution activities includes change propagation, impact analysis, inconsistency management, model refactoring, code generation, reverse engineering, version control, and traceability management [3]. Missing from this list is the integration and reuse of software design and domain knowledge. And yet, capturing both the experience of software design engineers and domain knowledge and then reusing such experience and knowledge would save time, effort, and cost of future development.

In this work, we organize software design knowledge into three types: design representation, design rationale, and design implementation. Design representation includes abstract descriptions of what a software system should do and how it should be done. Design representation includes software artifacts such as requirements, use cases, patterns, and design diagrams. Design rationale "is the explicit listing of decisions made during a design process and the reasons why those decisions were made" [4]. Design implementation includes the platform specific descriptions of the design representations (such as code and test plans). Considerable research focuses on knowledge reuse in design rationale and design implementation. While some knowledge reuse exists in design representation, such as software patterns, reuse of knowledge related to the syntactic and semantic rules governing the relationships among sub-structures of abstract design artifacts is practically non-existent.

We address the above three challenges in software evolution by combining model weaving with ontologies to facilitate the reuse of software design and domain knowledge, and the integration of multiple software modeling languages in the software evolution of abstract artifacts. Section 2 briefly reviews model weaving and ontologies. Section 3 introduces our initial steps towards combining model weaving and ontologies as a promising solution to these challenges. Section 4 reviews related literature. Section 5 concludes with the next steps in our research.

2 Model Weaving and Ontologies

Model weaving is a form of model transformation. Stated simply, model transformation involves transforming a source model to a target model. Model transformation is at the heart of a variety of techniques including forward engineering from models to code, refinement and refactoring of models, transformation between models, and reverse engineering from code to models. The OMG's Model Driven Architecture (MDA) [5] defines guidelines on model definition and transformation. Model weaving utilizes a weaving model that has typed links containing user-defined semantics to map between model elements [6]. Fig. 1 portrays an overview of model weaving. Weaving links specify the semantic relationships between source and target models above and beyond the one-to-one element matching of most model transformation approaches. The weaving model conforms to a predefined weaving metamodel that defines a variety of mapping capabilities. With model weaving, one element of a source model can be linked to a set of elements in the target model and

vice versa. Complex mappings such as n:1, 1:m, and n:m are possible as well as expressions such as equality, equivalence, non-equivalence, and generality via metamodel extensions and mapping expressions [7]. The weaving model is incorporated into a transformation program that performs the actual model transformation. Creating a weaving model is a semi-automated process in which many similarities among model elements can be identified automatically but manual refinement may be necessary.

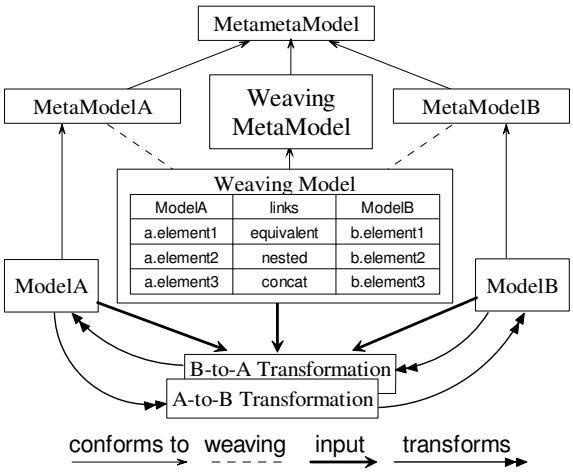


Fig. 1. Model weaving.

Model weaving offers three advantages over other model transformation techniques [6]: links in a weaving model are bi-directional whereas most model transformation techniques produce unidirectional transformations; transformation patterns associated with weaving links are more reusable than the structure dependent coding patterns of most model transformation approaches; and changes to source and target metamodels propagate through weaving links with fewer modifications to the weaving metamodel than in other model transformation techniques in which source and target model changes require changes in the model transformation program.

Utilized for several years in philosophy, linguistics, and artificial intelligence, ontologies are now a popular knowledge representation model in a variety of software development areas such as multi-agent systems, natural language processing, and information retrieval. An ontology consists of hierarchically arranged concepts, relationships among those concepts, and rules that govern those relationships. While no standard definition of ontology exists, a commonly accepted definition describes an ontology as a formal, explicit specification of a shared conceptualization [8], [9]. An ontology is, therefore, an abstract model of some area of knowledge used to share information regarding that knowledge area. It contains explicitly defined and generally understood concepts and constraints that are machine understandable.

An example of an ontology that we utilize in our research is the Ontology for Software Specification and Design (OSSD) Model [10]. A partial view of this model is given in Fig. 2. The OSSD model is an ontology of software design and specification knowledge that consists of hierarchically arranged software

development concepts, relationships, and rules. The OSSD Model integrates the structural and relationship knowledge acquired from multiple views of a software design. The graphical notations of the OSSD Model include: rounded rectangles representing classes interconnected via solid lines implying "Is-a" relations; and dashed lines representing properties that describe additional details regarding classes and conceptually link related classes. The OSSD Model utilizes ontological reasoning via rules associated with its properties to assist with both syntactic and semantic error detection among multiple design views.

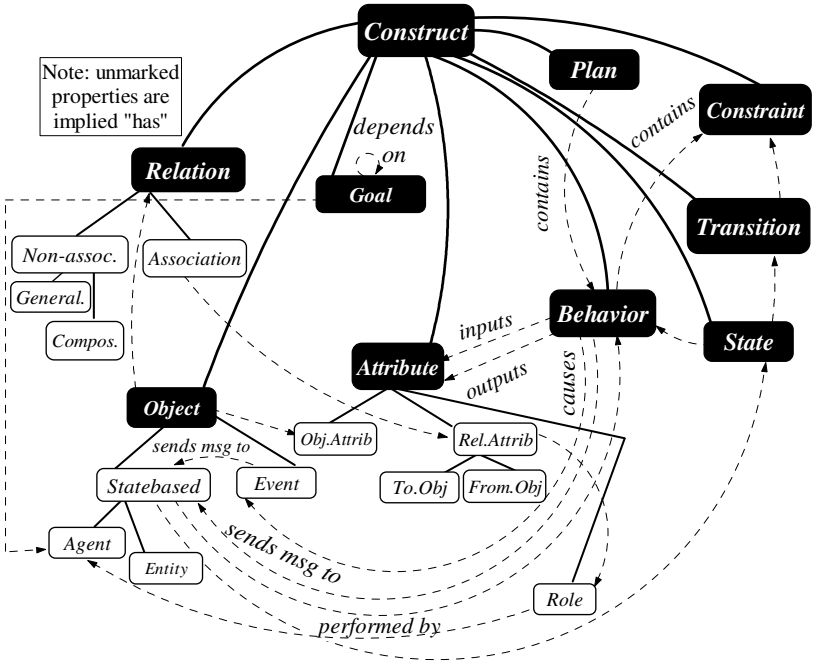


Fig. 2. Partial view of the OSSD Model.

In Section 3, we present a high-level view of our approach towards combining ontologies and model weaving to assist with evolving abstract artifacts.

3 Our Approach

Our approach facilitates the evolution of abstract software artifacts such as requirements models, encourages domain knowledge and software design knowledge reuse, and integrates software models specified using the same or different software modeling languages. Conceptually, we weave together ontological models representing both software design knowledge and domain knowledge as shown in Fig. 3. The Generic Design Model consists of properties, or rules, governing the relationships among software requirements and design constructs. The Generic Domain Model focuses on facts and rules within a given domain. We utilize the

OSSD Model as the Generic Design Model and are defining weaving models (Design WMs) to integrate the metamodels of other modeling languages with the Generic Design Model. We are developing the Generic Domain Model and associated

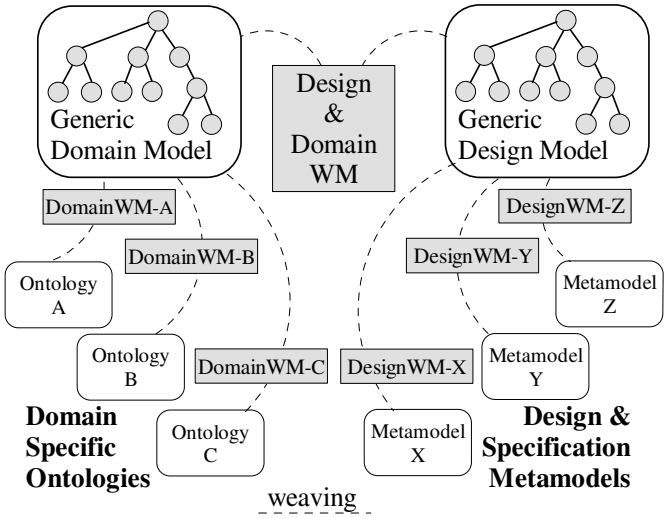


Fig. 3. Weaving Domain and Design Knowledge.

weaving models (Domain WMs) to integrate a variety of domain ontologies. We are developing the Design and Domain Weaving Model to integrate generic software domain and design knowledge. Our approach facilitates knowledge reuse by applying knowledge acquired during processing of one application to subsequent applications via our generic ontological models. We utilize the ontological models shown in Fig. 3 to store initial requirements and design models that are later accessed during the evolution of those models thereby facilitating knowledge reuse and multi-language integration.

As a demonstration of our approach, we consider the evolution of an airport security software system to incorporate new requirements such as those discussed in the case study presented for analysis at the 14th Monterey Workshop [11]. In our approach, the initial requirements for the airport security system, possibly specified using a modeling language such as the Unified Modeling Language (UML) [12], would be transformed into two ontological representations, one for design knowledge and one for domain knowledge as shown in Fig. 4 via the Airport Security Design and Airport Security Ontology respectively. Knowledge from these two ontologies would be woven into instances of the Generic Models. New requirements such as a ban on carrying liquids in passenger luggage would be specified in UML or some other modeling language and woven into the existing ontological models for the airport security system. Our approach enables the creation and reuse of rules, such as rule21 that controls the relationship between domain constructs “a” and “b” or rule32 that controls the relationship between design constructs “d” and “e”. Now, suppose that rule21 concerns the screening of passenger luggage (represented by node a) for guns (represented by node b) and a new requirement is created to add liquids (represented

by node c) to the screening. A new domain rule would be added between nodes a and c. This new rule would be woven into the design model and included as a new rule affecting passengers (represented by node d) and their luggage (represented by node

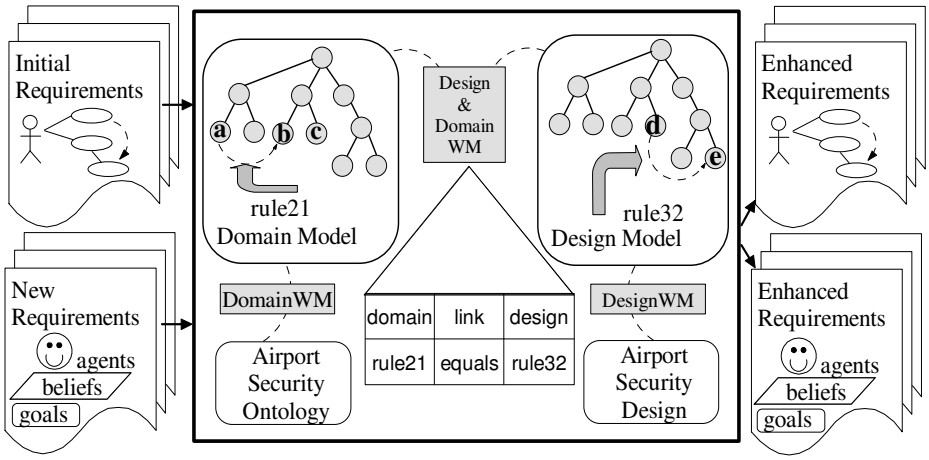


Fig. 4. Our Approach.

e). The appropriate DesignWM would then be utilized to generate the final product as a model containing the enhanced requirements specified in either the original or new modeling language.

4 Related Research

Software evolution is a broad research area. We narrow our research review to the evolution of abstract software artifacts and specifically those incorporating ontologies, model weaving, or knowledge reuse.

Several ontological solutions are emerging to address software evolution. One approach assists software maintenance processes by using ontologies and automated reasoning, via description logics, to represent heterogeneous software maintenance artifacts [13]. Their work creates separate ontologies for source code and documentation (such as requirements and design) and maps between them providing query and reasoning capabilities. Their work provides considerable analysis capabilities but does not produce evolved artifacts as our approach does. Much of the related work utilizing ontologies in software maintenance focuses on representing software maintenance knowledge. For example, a software maintenance ontology [14] provides a unifying framework for software evolution tool interaction. This ontology consists of high-level maintenance concepts such as software system, modification processes, computer science skills, organizational structure, and application specific knowledge. Their software system components include artifacts such as requirements specification and design specification, while our ontology addresses software development artifacts at a more detailed level, such as object, agents, behavior, and goals.

Current software reuse research focuses on representing and retrieving software artifacts such as code, patterns, components, and experience. While patterns share design knowledge they do not facilitate reasoning with that knowledge nor address domain knowledge reuse. The KOntoR approach [15] provides both domain knowledge reuse and reasoning capabilities by storing software artifacts in a metadata repository and utilizing ontologies to represent both software design and domain knowledge. While the KOntoR approach also processes software artifacts specified in variety of formats, its reuse does not incorporate rule knowledge concerning the relationships among software design constructs. REBUILDER UML [16] facilitates reuse of software design knowledge utilizing ontologies and Case-Based Reasoning (CBR). This tool combines UML class diagrams with domain ontologies to provide users with a software design knowledge library of problem, solution, and outcome cases. It differs from our approach because it focuses on one software modeling language, knowledge reuse only at the object or class diagram level, and uses ontologies to represent only domain knowledge.

5 Future Work

We presented our initial steps towards combining ontologies with model weaving to facilitate the evolution of abstract software artifacts such as requirements models, encourage domain knowledge and software design knowledge reuse, and integrate software models specified using the same or different software modeling languages. These steps represent a part of our ongoing work to implement a system called the Evolution Weaver

References

1. Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R., Jazayeri, M., Challenges in Software Evolution, International Workshop on Principles of Software Evolution, Portugal (2005)

2. Berrisford, G., Why IT Veterans are Sceptical about MDA, 2nd European Workshop on Model Driven Architecture (MDA), UK (2004) 125-135

3. Mens, T., Van Der Straeten, R., On the Use of Formal Techniques to Support Model Evolution, Technical Report TR-CCTC/DI-35, Departamento Informatica, Universidade do Minho, Portugal (2005) 67-98

4. Jarczyk, A., Loeffler, P., Shipman, I.F., Design Rationale for Software Engineering: A Survey, 25th Annual IEEE Computer Society Hawaii Conference on System Sciences (1992) 577-586

5. Object Management Group, Model Driven Architecture, V1.0.1 (2003)

6. Del Fabro, M., Jouault, F., Model Transformation and Weaving in the AMMA Platform, Generative and Transformational Techniques in Software Engineering, Portugal (2005) 71-77

7. Del Fabro, M., Bezivin, J., Valduriez, P., Weaving Models with the Eclipse AMW Plugin, Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany (2006)

8. Gruber, T., A Translation Approach to Portable Ontology Specifications, Knowledge Systems Laboratory Technical Report KSL 92-71, Stanford University (1992) Revised (1993)

9. Borst, W., Construction of Engineering Ontologies, Ph.D Dissertation, University of Tweenty, Enschede (1997)

10. Hoss, A., Carver, D., Ontological Approach to Improving Design Quality, IEEE Aerospace Conference, MT (2006)

11. The 14th Monterey Workshop, Case Study: Air Traveling Requirements Updated, Workshop on Innovations for Requirements Analysis: From Stakeholders Needs to Formal Designs, CA (2007)

12. Object Management Group, Unified Modeling Language (UML) 2.0 (2003)

13. Witte, R., Zhang, Y, Rilling, J., Empowering Software Maintainers with Semantic Web Technologies, to appear at the 4th European Semantic Web Conference, Austria (2007)

14. Anquetil, N., de Oliveira, K., Dias, M., Software Maintenance Ontology, Ontologies for Software Engineering and Software Technology, Springer Berlin Heidelberg New York (2006)

15. Happel, H., Korthaus, A., Seedorf, S., Tomczyk, P., KOntoR: An Ontology-enabled Approach to Software Reuse, 18th International Conference on Software Engineering and Knowledge Engineering, CA, Jul. (2006) 329-344

16. Gomes, P., Leitao, A., A Tool for Management and Reuse of Software Design Knowledge, 15th International Conference on Knowledge Engineering and Knowledge Management, Vol. 4248, Lecture Notes in Computer Science (2006) 381-388

Text Classification and Machine Learning Support for Requirements Analysis Using Blogs

Douglas S. Lange

Space and Naval Warfare Systems Center
San Diego, CA 92152
doug.lange@navy.mil

Abstract. Text classification and machine learning technologies are being investigated for use in supporting knowledge management requirements in military command centers. Military communities of interest are beginning to use blogs and related tools for information sharing, providing a comparable environment to the use of blogs for system requirement discussions. This paper describes the work in the area being performed under the Personalized Assistant that Learns program sponsored by the Defense Advanced Research Projects Agency. Comparisons are then made to how the technology could provide similar capabilities for a requirements analysis environment.

1 Introduction

The United States Military has adopted several network based communications mechanisms. During the second Gulf War, *chat* was an important method of communications, reducing the need for voice circuits. *E-mail* protocols have been used heavily since the early '90s for longer more structured messages in the place of old teletype methods. Now, *blogs* and *wikis* have come into use for knowledge sharing purposes [1].

The U.S. Strategic Command has developed and uses heavily a capability that can best be described as a hybrid of wikis and blogs. The Strategic Knowledge Integration Web (SKIWeb) allows users to post information about key events, and allows other users to add comments and edit the information. Events can be linked to other events, and lists of events are used to provide key information to various communities of interest [2]. While SKIWeb is structured differently than most blogging capabilities, the information within it can, with small transformations, be represented as being structured exactly like a collection of blogs.

With sponsorship by the Defense Advanced Research Projects Agency (DARPA), the Space and Naval Warfare Systems Center (SSC) along with SRI International is working to transition machine learning technology into both SKIWeb and a blogging capability being developed for the U.S. Navy's Composeable FORCEnet (CFn). It is envisioned that the learning technology can not only aid the bloggers, reducing the labor costs of publishing information, but can also extract information for other

purposes. It is this second feature that is most closely aligned with the goal of extracting software requirements from blogs.

2 PAL Blogs

Machine learning from the DARPA Personalized Assistant that Learns (PAL) program is being used in several ways in conjunction with blogs. These approaches will help both those who publish information on blogs and those who subscribe to receive the information over Really Simple Syndication (RSS) feeds. Figure 1 depicts the CFn PAL Blog design.

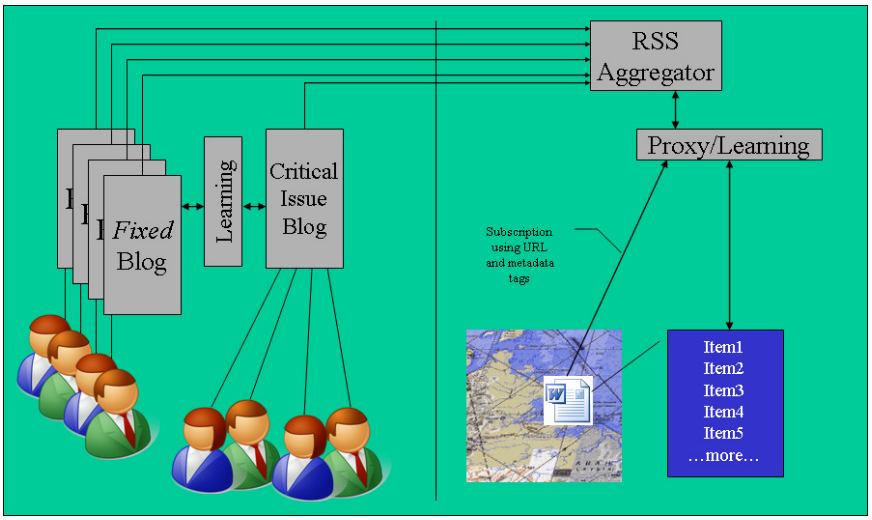


Fig. 1 PAL Blog

2.1 Learning on the Publishing Side

Two PAL technologies are being applied to the publishing side in the CFn capability. The first is text classification. Various learning techniques can be applied to text to help map the topics found in a corpus. Various algorithms can be used [3] and the selection can depend on the characteristics of the text, and the mode by which the classifier is to be trained and used.

Text classifiers occupy the *Learning* module in Fig. 1. By subscribing to the blog of a user, text classifiers can determine what topics the user writes about. If a user frequently reports the status of aircraft in his/her blog, a model of the writer's interests can reflect that. Further, through the use of intelligent search and indexing that employs the same topic mapping capabilities, PAL may find new email, documents,

chat, or other information sources that have new information on aircraft status and suggest to the user that the information be added to the blog. In Section 3, it will be argued that this capability can be useful in defining system or software requirements from blogs.

A second technology that is being utilized for the PAL Blog provides *task learning* [4]. Another approach to helping the user publish is for PAL to learn and generalize common tasks for the user. If the user frequently gets email about the status of aircraft, he or she may typically choose to do additional research before publishing the results. Perhaps a database of parts needs to be queried and a decision aid to calculate delivery times must be run. The PAL task learning technology allows a user to teach PAL that when such email arrives in the future, PAL is to go through those steps and report the results in a particular manner on the blog, thereby relieving the user of the need to perform the task. PAL. This technology is not directly applicable to an effort to use natural language tools such as blogs as requirements sources, but task learning itself can be a means to requirements gathering by using the task models that are generated as representative of the capabilities required.

2.2 Learning on the Subscription Side

On the subscription side, the goal of learning is to predict what information the user would like to see. PAL is being used to observe the reading habits of users and suggest RSS feeds to subscribe to, and even which entries from a feed to treat with higher priority. This is being done with text classification methods, mapping of topics, and even social networking clues such as which bloggers provide more authoritative information.

2.3 iLink

The third technique being applied to blogging activities is social network analysis. The iLink capability, developed by SRI International as part of the PAL program, learns to attribute different levels of expertise based on how subscribers judge the contributions of publishers. The way iLink is structured, when a user poses a question through iLink it is distributed to those who are known to have some expertise in the area. Recipients are able to answer the question or forward it to others who they feel might know the answer. When answers do come, those who have provided useful information and those who referred the question to them have their scores raised. Those who provide poor information or cannot answer have their scores lowered. In this way, expertise is determined by the quality of information rather than by simple claims in social network metadata.

3. Applications for Requirements Analysis

If we consider that the text of individual blogs contains information important to the requirements of a system being developed, then the use of machine learning text classification tools may provide a means to organize the statements, record arguments for and against a capability, and even provide some sense of priority.

The text classification that is being done for PAL Blog on the both the publishing and subscription sides groups blog entries based on the topics being discussed by finding similarities in the words being used. Entries that are discussing the same topics may be the essence of a discussion about a particular capability, and perhaps can be divided into arguments for different ways of providing the capability in question.

Through social network analysis, as blog entries are analyzed for good requirement content, judgments made about the quality of the input can be made by engineers and used to learn the level of authority that should be attributed to individual authors both through the judgment of the engineers and by the level of agreement with those that are already judged to be authoritative.

4. Results from Experiments with Blogs

During the summer of 2007, there will be two experiments done in analyzing blogs. The first will be at the U.S. Strategic Command. Classification of events and the comments made by bloggers on them will be attempted with operational data. The second is in a web portal called PlatoonLeader where U.S. Army officers share information about how to handle particular requirements of leading a platoon. These comments are not too different than discussions about how a system should be developed to provide useful capabilities. Both the topic mapping and the authority scoring tools are being used for this application. Full results from both of these experiments should be available in September of 2007.

References

1. Seymour, G. and Cowen, M., “A Review of Team Collaboration Tools Used in the Military and Government”, www.onr.navy.mil/sci_tech/34/341/docs/cki_review_team_collaboration.doc, Accessed on 6 April 2007.
2. Boland, R., “Network Centricity Requires More than Circuits and Wires”, Signal, Vol. 61, No. 1, September 2006.
3. Lange, D., “Boot Camp for Cognitive Systems: A Model for Preparing Systems with Machine Learning For Deployment”, Ph.D. Dissertation, Naval Postgraduate School, Monterey, CA, March 2007.
4. Conley, K, and Carpenter, J., “Towel: Towards an Intelligent To-Do List”, Technical Report, SRI International, Menlo Park, CA, 2006.

Requirements Documents and Opportunities for Natural Language Processing

Barbara Paech[1]

University of Heidelberg, Im Neuenheimer Feld 326, 69120 Heidelberg
paech@informatik.uni-heidelberg.de

Abstract. In this paper it is argued that the application opportunities of natural language processing to requirements depend strongly on the quality of the requirements documents. This is illustrated through examples from quality requirements.

1 Introduction

Natural language processing (NLP) is concerned with NL understanding and NL generation [3]. This can be applied to text and speech. Depending on the formality of the text or speech different problems arise and different mechanisms apply. In Requirements Engineering (RE) typically there is a wide variety of communication and documents characterized also by a wide range of formality. In order to identify possibilities for NLP in RE it is therefore important to determine the relevant levels of formality. In this position paper I propose 4 major levels of formality (section 1). In section 2 I discuss 4 major applications of NLP which help to elicit, document, evaluate and manage RE information on these 4 levels. This is illustrated by concentrating on an important subset of requirements, namely quality requirement (QR), which have been the focus of some of my research during the last years [4,2,1]. I conclude with some remarks on future research questions.

[1] Currently on Sabbatical at University of New South Wales, School of Information Systems, Technology and Management, Sydney NSW 2052

2 Requirements Documents

During RE several documents are used to capture information on the features and the quality of the system to be developed. Often at least two different kinds of documents are distinguished, namely a more informal one capturing stakeholder wishes which might be ambiguous, conflicting and imprecise and a formal one capturing the consolidated requirements which can be used as input for design, testing and management. The former is sometimes called requirements or early requirements in contrast to specification or late requirements for the latter.

Table 1: Requirements Information Characteristics

Document	Form	Syntax	Consistency	Ambiguity	Description	QR example
Wishes	Tacit or spoken	Natural	Inconsistent	Implicit and explicit	Wishes of the stakeholders in their heads, some might not even be known to them	I do not want any security problems on airplanes. I do not want so much detailed work for the screening personnel.
Raw requirements	Spoken or written	Natural	Inconsistent	Implicit and explicit	Wishes of the stakeholders explicitly uttered during RE	We need new ideas for screening. The screening should not make people tired too much.
Consolidated requirements	Written (Text or diagram)	Natural or restricted text	Logically consistent	Explicit	Wishes of the stakeholders consolidated so that all conflicts and ambiguities are explicit	We need to screen for A…L. After screening for 1 hour the alertness of the screeners should be still 80% .
Specification	Written (Text or diagram)	Natural or restricted	Logically consistent without conflict	Explicit	Agreed, reliable input for design, test and management	The system should screen for A…G. The system should only use colors … to avoid stress for the screeners.

In the following I propose to distinguish 4 levels according to the following attributes

- Form: tacit, spoken or written
- Syntax: natural, restricted
- Consistency: realizable (that means without conflict), logically consistent (but may be not realizable), inconsistent (in the sense of not logically consistent)
- Ambiguity: explicit (explicitly mentioning several possible realizations), implicit

These attributes are distinguished because they are important for NLP: NLP can detect to some degree that there might be tacit knowledge, but it can only process spoken or written information. Clearly, restricted syntax alleviates NLP. NLP can detect to some degree logical inconsistencies, but realizability can only be assessed with a lot of domain knowledge which is typically not available today for NLP tools. Implicit ambiguity (which is just inherent in NL) makes NL understanding difficult, while explicit ambiguity is no problem for NLP. Note that in contrast to e.g. [5], I do not treat completeness. Formal completeness in the sense that e.g. a distinction of cases has to be complete is comprised by ambiguity, because a missing case would entail an implicit ambiguity.

Table 1 describes the 4 kinds of information and their characteristics. Basically I have separated the early requirements into *whishes* – that is information which is mainly in the head of the stakeholders (but nevertheless the major threshold for stakeholder satisfaction) - and *raw requirements* – that is information which is uttered explicitly by the stakeholders. Furthermore I have separated the late requirements into *consolidated requirements* – that is information which is consolidated so that all stakeholders have the same interpretation of it (as far as this is possible at all between humans), but still might not agree on it, - and *specification* – that is information on which all stakeholders have agreed and which therefore can be used in the development as a reliable basis for design, test and management. Nowadays a specification could also be part of the system itself, e.g. in agent or ad-hoc systems where the components explicitly carry their specification with them to allow its analysis at run-time.

Clearly wishes and raw requirements will always be part of a development project (assuming that design does not start without anything written down), while it depends on the effort during RE whether consolidated requirements or a specification are created.

The example QR are related to the workshop case study. The wishes represent ideas of the stakeholders which might be uttered explicitly or not. The raw requirements make these wishes explicit, typically in a different form, as stakeholders often do not formulate their true wishes. In particular, instead of the QR (e.g., no security problems) a specific functional requirement (e.g., new screening measures) might be given. This transition is often made not consciously. It can occur on all information levels and rationale for it should be captured. Consolidation in the area of QR means in particular providing detailed metrics. So in order to consolidate the 2 raw statements, specific screening

measures (here symbolized by A…L) and specific metrics for "being tired too much" have to be provided. Clearly, these statements still might be conflicting. The specification should provide detailed targets for the design. To detect and solve potential conflicts, each of the measures A…L has to be investigated to find out whether their screening individually or in sum invalidates the alertness requirement. In this case it is assumed that on the one hand the number of the screening measures has to be reduced and on the other hand specific user interface guidelines are needed to support the alertness when screening A…G. Clearly, the transition from raw requirements to specification involves a lot of creativity and it will be the case that to solve conflicts new raw requirements are needed and consolidated requirements have to be discarded, thus making the consolidation step of the discarded requirements seemingly superfluous. Still, it is important to first get a clear understanding of the requirements before conflicts can be detected and resolved. However, it also means that the consolidation steps and the conflict resolution steps typically will only be applied to subsets of requirements so that the whole specification is achieved through an intertwinement of consolidation and conflict resolution.

3 NLP Opportunities

So what can NLP do for RE? I distinguish 4 typical applications

- Inconsistency and Ambiguity Detection (IAD): this can be used to create consolidated requirements out of raw requirements.
- Translation: this can be used to translate between information in different syntax. So e.g. natural raw or consolidated requirements could be translated to restricted use case text or more formal models like state charts. But also from a restricted syntax (e.g. diagrams or formal languages) natural language syntax could be generated to make it better understandable for some stakeholders.
- Question-Answering System (QAS): this can be used to elicit more information on each level. So e.g. given raw requirements the NLP tool can ask questions to provoke the stakeholders to make their tacit wishes explicit (thus improving the raw requirements), but also even to provoke new wishes (thus improving the wishes).
- Information Analysis: this can be used as a pre-step to translation or question-answering, but also on its own to make explicit specific characteristics of the information. So e.g. the NLP tool could look for patterns in order to identify or compare other information sources relevant to the given information or in order to provide guidelines for similar information.

All of them require a syntactic, semantic and pragmatic understanding of the information [3].

The first two applications support the *documentation* of requirements the last two support the *elicitation* of requirements. All can support *evaluation*: IAD and information analysis obviously target information quality. Translation supports evaluation as re-phrasing always provides a new perspective. Clearly formalization helps to detect quality problems, but also the other way round supports evaluation as new stakeholders can be enabled to evaluate the formal information. Similarly, QAS puts the information into a new perspective which helps to detect quality problems. Furthermore, all support *management*, as they provide explicit traces between information they process and the outcome of the processing. In addition, QAS typically provides traces to rationale of requirements, information analysis provides traces to other information and between the requirements on one level and IAD provides traces to quality problems.

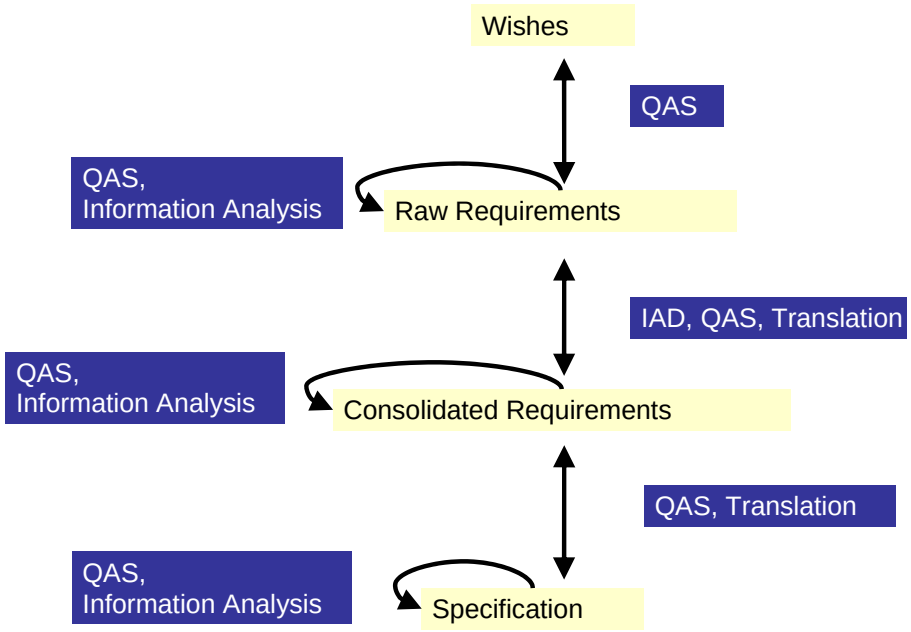


Figure 1: Application of NLP

As shown in Figure 1 the applications have different emphasis wrt. 4 levels: IAD is only useful at the level of raw requirements (this also includes the case of adding information to the consolidated requirements, bcause the new information should be viewed as raw requirements before integration). QAS can be used to improve each of the levels, where wishes can only be questioned on the basis of the raw requirements. Translation can be used between different syntax and information analysis can be used on speech or text. Clearly, information analysis and translation are more difficult in the

presence of implicit ambiguities and logical inconsistencies. Therefore it seems helpful to first apply IAD. On the other hand information analysis can also help to reveal information which is important for IAD, e.g. domain knowledge.

It is important to point out that in any case NLP for RE involves humans. For QAS this is trivial. The information provided by information analysis has to be evaluated by humans wrt. their usefulness. During IAD humans are needed to resolve the detected ambiguities and inconsistencies. When consolidated requirements are translated to specifications, humans are needed for the agreement step. And also for the translation of raw requirements to consolidated requirements typically a lot of creativity is needed (as mentioned for the QR examples above).

Just to illustrate the NLP opportunities discussed above let us look again at the workshop examples. Given the general topic airport security QAS could stimulate the stakeholders to utter the wishes as raw requirements (as they are given in the workshop case study). In particular it could offer different scenarios relevant to security problems. A human requirements engineer is needed to drive and monitor this process. The raw requirements can be analyzed to detect patterns, e.g. topics which are particular important for specific stakeholders, or related information which should be considered in the discussion. The requirements engineer should decide whether further uses of QAS given the additional information are helpful. Furthermore, translation into a more restricted language for the raw requirements might be useful to enable more efficient processing. Then s/he can decide to detect quality problems (like the missing metrics in the QR) though IAD. The solution of the problems involves a lot of human creativity where QAS and information analysis could be used repeatedly as above to elicit and improve new raw requirements, and IAD to detect problems with these new requirements. After the consolidation a target language for the specification is chosen. As illustrated for the QR, in the translation of the consolidated requirements to the specification again a lot human creativity is needed, in particular to investigate and solve the conflicts. NLP can be used to automate translation wherever possible, so that the only the creative work is left to the humans. The feasibility of this support clearly depends very much of the languages used for consolidated requirements and specification. The more restricted the easier to process for the computer, but often the more difficult for humans. As for the step from raw to consolidated requirements QAS, information analysis and IAD can help to treat new requirements.

4 Conclusion

The discussion above shows that the wide variety of information and communication and formality levels in RE embodies many opportunities for NLP. But it also shows that it is not easy to give a compact and stringent view on these opportunities. With the proposed classification into 4 kinds of information processed by 4 major NLP applications I have

tried to provide such a view. Based on this view and my limited knowledge of current NLP approaches (in RE) I conclude that the following research areas should be emphasized:

- A lot of NLP for RE research has so far concentrated on IAD and translation, in particular in the sense of formalization. As shown above there are also many opportunities to support the elicitation stage through NLP. These should be explored
- The efficiency of NLP support for RE depends crucially on the languages used to formulate the requirements. Much research has focused on either natural or very formal languages. Both are not adequate in most cases for RE, because the concepts used in the information are on the one hand specific for the application domain (such as e.g. airport security) and on the other hand specific for how the information is to be used in the development process (e.g. QR need to be documented differently from functional requirements (FR) as they are treated differently in the development process. Typically QR are very relevant for architecture decisions while the FR shape the system functions). More research is needed to identify languages which capture these concepts and their treatment in the development process nicely, and at the same time are understandable for the relevant stakeholders and easily amenable to NLP.

References

1. Herrmann, A. Paech, B. “Quality Misuse”, Int. Workshop REFSQ’05, Essener Informatik Berichte, pp. 193-199 2005.
2. Herrmann, A. Paech, B. “MOQARE: Misuse-oriented Quality Requirements Engineering”, Requirements Engineering Journal, Springer-Verlag, accepted for publication.
3. Jurafsky, D. Martin, J.H. Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition, Draft 2007, Introduction, http://www.cs.colorado.edu/%7Emartin/SLP/Updates/1.pdf
4. Paech, B. Kerkow, D. “Non-Functional Requirements Engineering - Quality is essential”, Invited paper, Int. Workshop REFSQ'04, Riga, pp. 237-250, 2004.
5. Pohl, K.: The Three Dimensions of Requirements Engineering; Proc. of the CAiSE Conference, 8-11 June, Paris, Springer-Verlag, 1993.

Model-Based Requirements Specification and Tracking for Service Oriented Architecture (SOA) Based Systems.

John Salasin, Ph.D.
Visiting Researcher
National Institute of Standards and Technology

1 Requirements and Models

Requirements, whether expressed as a diagram or as text that captures a discussion about needed capabilities, are the first of a family of models that evolves with the system. Typically, these models (of requirements) start with "back of the envelope" guesses and ending up with a fully elaborated model (the system).

Several requirements need to be satisfied for model specification and elaboration to be successful. First, for Service Oriented Architecture based systems, it is a basic tenet that business managers, as opposed to information technology specialists, define major aspects of the system, which map to the business processes of the organization. This helps ensure that the information system serves the business goals of the organization. This is one reason why the Object management Group's (OMG) Model Driven Architecture (MDA) approach begins with a Business Process Modeling Notation (BPMN) -- a graphical notation that depicts the steps in a business process. BPMN has been designed so that business managers should be able to easily read and understand a BPMN diagram. This implies that any (initial) formal specification of requirements, which requires an understanding of formal logic(s) and takes significant time for a non-mathematician to learn, will not be suitable for defining requirements for business-driven systems.

Second, the early models of requirements should be as inexpensive for an organization to produce as a back of the envelope diagram. This makes the use of textual descriptions processed by natural language understanding (NLU) tools improbable. NLU is based on an ontology that describes the concepts involved in the business/system and relationships (e.g., of causality, correlation, containment) among these concepts. Ontologies are currently very expensive and time consuming to construct over a broad domain that incorporates multiple contexts where the concepts and/or their relationships differ. Some systems, such as the Semantic Network Processing System (SNePS)[1] and Cyc[2] address the problem by partitioning the ontology according to context (e.g., microtheories in CyC), where the logic must be consistent within a context but not between contexts. Judgments about which context(s) to use in a given situation require someone intimately familiar with both the language and the domain(s).

The time, cost, and skill required to build and use ontologies is not an issue in simple domains. The Army, for example, has used artificial intelligence to lead visitors through its recruiting Web site – replacing live recruiters with online chat.[3]

Gary Bishop, deputy director of the Strategic Outreach Directorate of the Army's Accessions Command, stated that "About 60 percent of the questions were always the same ones." While the article does not discus the technology used, a model of the frequency of key words (or their synonyms) in a set of the most typical questions should be sufficient to point to the correct answer.

2 Models and Metrics

Models express the "global invariants" of development. They must have elements that are consistent, yet be adaptable to express different levels of refinement and accuracy as design/implementation decisions are made and additional data collected. Different sets of metrics from families of similar models are needed to illuminate different characteristics of the system – from effectiveness and efficiency at supporting the business mission (e.g., impact on business performance versus resources required) to technical concerns (e.g., response time, throughput, computer and communication resource requirements). Measures based on the models provide metrics that can be used to express the importance/significance of system characteristics, e.g., requirements. As the system specification (and models) become more refined, they can ce used to assess the extent to which the requirements are being (or will be) fulfilled.

We consider requirements/models throughout a project's evolution.

- The earliest stages focus on concept development and obtaining management support. The functions of the system and interactions with other organizations are modeled. These (rough) models are used to estimate the scope of the services provided in terms of the number and types of consumers – from a single program to the public at-large.
- During architecture and construction, a number of SOA components are specified and prototyped/implemented. Requirements (models) specify, and, if the system is successful, predict, the impact of the IT initiative on business results. They specify or assess specific characteristics of the system and its harmonization with processes and systems in other parts of the organization.
- During operation, the requirements / models posited in previous stages can be refined and validated. The focus is on capturing (rather than estimating) key elements of performance that directly relate to the IT initiative. Models can estimate (or measure) robustness, actual usage of Services or Service components, usability, and the extent of ongoing governance – assessing whether requirements are satisfied and, possibly, resulting in changes to original requirements.
- As the system evolves, we can collect data to validate requirements/models specifying the ease with which a system can be modified and expanded to include additional organizational units in the enterprise.

3 What's Hard about SOA (unique features)

While SOA is a current fad in software development, it appears to provide the approach most likely to succeed in facilitating the development of enterprise-wide systems that operate across heterogeneous platforms. In the real world, SOA systems evolve. They start by providing functionality to a small number of organizational units and, if successful, "metastasize" to the rest of the enterprise.

SOA is not a discrete technology or language, but a mixture of an architectural style and development approach that is intended to keep automated system design synchronized with business processes and to allow business area managers, rather than computer scientists, to define services and rules for their interaction.[4] Useful requirements and models for SOA must, therefore, go beyond static (or dynamic) code analysis and technical performance to include factors related to information utility and acceptance of the system by the organization(s) participating in an enterprise. To be objective and repeatable, such metrics need to be based on technical characteristics of the system (e.g., notations used to express both business processes and system architecture).

Business managers, as opposed to IT specialists, are expected to define major aspects of system. The system requirements, reflected in system models, need to specify the ease with which this can be done.

4 SOA requirements

SOA requirements, and the models used to express them, must address factors related to the:

- Ease with which a system can be modified or expanded by business, rather than IT, specialists;
- Extent to which the system provides all functionality required by all groups of customers (Comprehensiveness);
- Factors that facilitate the spread of a system (or system-of-systems or EA) across an organization;
- Capability of the organization responsible for a service (or an enterprise) to specify, monitor and enforce all policies of concern, and;
- Utility, or value added by the System to the organization.

Some useful requirements, whose fulfillment can be assessed by models, include the following.

4.1 Ease with which a system can be modified or expanded

- Extent to which the notations used to specify the system enable ease of modification – including support to the co-evolution of business and technical processes.

- The percentage of system design elements (e.g., services and process flows) that are explicitly linked to business process elements;
- The percent of operations that can be expressed in simple scripting language (e.g., by business managers vs. programmers).
- The effort to make a change (e.g., Hours to make corresponding changes to business and service specifications (e.g., substitution of a data item).
- Ease of changing the systems to serve different customers in different organizations. Potential requirements/measures include:
    - The effort required to make typical changes in, for example, services/processes, data stores, service orchestration, and to check on policy consistency / applicability when system changes are made;
    - Number of models available to assess critical aspects of the system including factors related to, e.g.,: scalability, response time, resource use, recovery from anomalies, overhead for rule monitoring and enforcement.

4.2 Extent to which the system provides all functionality required

- Ability to trigger businesses processes in response to detected events (both possibly outside the SOA)? Specific requirements / measures might include the percent of anticipated external “effector components” that the system can “talk to”;
- The amount of effort involved in writing triggering code. Is there a simple scripting language to specify triggering conditions and actions? How long does it take to learn?
- The percent of a supported task that can be accomplished with single sign-on;
- The percentage of services/functions that need to collaborate which are integrated in the system;

4.3 Factors that facilitate the adoption of a system across an enterprise include:

- The “fit” of system components (services) to higher levels of organizational structure (e.g., the percentage of system services that map directly to a defined organizational function or unit). While most services/SOAs will be developed incrementally (e.g., to support one functional unit at a time), effectiveness and efficiency require that the services are consistent with an overall organizational framework.
- The “two way” potential of the SOA system for incorporating (reusing) existing components and for the larger organization reusing components developed for the SOA. The first item increases an organization’s confidence, since they trust their own components, while the second provides some immediate benefits to organizational components not participating in pilot studies.
- A solid technology infrastructure, or computer engineering considerations that are usually not visible to customers under normal operations. These include measurements, or estimates, related to, e.g.:

- Fault detection and remediation;
- Robustness/Reliability -- including system-wide backup and recovery;
- Performance and scalability.

4.5 Capability of the organization responsible for an enterprise to specify, monitor and enforce all policies

- The percent of policies that can be easily specified monitored and enforced.
- The time required to define and implement triggers to provide alerts for (business) policy violations
- The expected effort and level of skill required to develop rules for specifying and enforcing policies related to allowed, e.g.:
  - Workflows, process sequences, pre- and post-conditions

4.6 Utility, or value added by the System

- The estimated cost to create an integrated business picture appropriate to each type of customer;
- Changes in delay (information collection to use) and the estimated value of more rapid and less expensive access;
- Increased scope of accessible information (e.g., cutting across stovepipe systems) – possibly measured as the percent of critical information elements needed for a typical decision that the system provides;
- Financial measures for producing products/services as measured by, e.g.,
  - An increase in the number/percent of missions supported by automated data collection, analysis, and reporting;
  - A change in the number/percent of program elements meeting (or expected to meet) cost/performance objectives.
  - Measures of the system's ability to provide information for improving management, and;
  - The time/training required for customer(s) to learn tasks.
- Total Cost of Ownership including the estimated range of time/effort to :
  - change policy rules, services, and service orchestration,
  - add various types of data sources,
  - trigger new reports.

4.7 Significance

This work is technically significant for 3 main reasons.

Service Oriented Architectures requirements (and their assessment) requires consideration of technology factors' themselves (e.g., with respect to response time) and their relationship to the organization (e.g., the ease of mapping business process descriptions to SOA descriptions; the ease of scripting rules, and; the ease with which system processing can be changed to account for changes in the business process).

We attempt to define all requirements (e.g., ease of use, flexibility) in terms of system characteristics that can be modeled and measured. The system is defined by a set of evolving models.

Since we begin expressing requirements with coarse grained conceptual models, measurement of these models' characteristics can indicate areas where the system might not meet some aspects of its requirements, allowing early corrective action. More refined models allow us to track the success of corrective actions.

---

[1] [S. C. Shapiro. Sneps: A logic for natural language understanding and commonsense reasoning. In Iw anska and S. C. Shapiro, editors, Natural Language Processing and Knowledge Representation: Language for Knowledge and Knowledge for Language, pages 175--195. AAAI Press/ MIT Press, Menlo Park, CA, 2000.]

[2] [D.B. Lenat and R.V. Guha, Building Large Knowledge-Based Systems (Addison-Wesley, Reading, MA, 1990]

[3] Government Computer News, "'Star' power", William Jackson, 02/19/07

[4] "Service-Oriented Architecture expands the vision of Web services, Part 1: Characteristics of Service-Oriented Architecture", Mark Colan (mcolan@us.ibm.com), IBM Corporation http://www-128.ibm.com/developerworks/library/ws-soaintro.html

Empirical Indicators for Very Early Functional Complexity Estimation on Data Intensive Information Systems

Pedro Salvetto[1], Juan Carlos Nogueira[1] and Julio Fernández[2]

[1]Information Systems Research Lab Faculty of Engineering Universidad ORT Uruguay
[2]Universidad ORT Uruguay
Cuareim 1451, ZIP Code 11100, Montevideo, Uruguay
{salvetto,nogueira,fernandez_j}@ort.edu.uy

Abstract. Intervention of expert judgment, along with intrinsic variability of estimation and development processes, allows for large gaps between optimistic and pessimistic estimations. In this work we study a sample of software development projects for data-strong management information systems. We found an indicator of functional complexity that can be used at very early stages to assess the effort required for developing the system. This early measure facilitates the communication and the negotiation during the functional requirement phase because it is objective and meaningful for end users.

Keywords: Empirical Complexity Indicators, Empirical Estimation Models, Formal Models

1 Motivation

Despite many similarities with ancient engineering and architecture disciplines, software engineering has not reached the same level of formalism [7,15,17,26]. The lack of early, automatically collected, and independent from human expertise metrics for functional requirement complexity is one of the drawbacks that contribute to retain software engineering in its primitive stage.
Such metrics could be useful in estimating time, effort, risk, and keeping track of the project development. Moreover, the metrics could improve the communication between users and developers providing a transparent tool to assess the cost of requirements, changes, and maintenance along the life cycle [3,9,10,13,15,26].
Industry, academy and society in general are asking for software development to evolve away from the artisan's paradigm, to professionalize and to become more predictable. One of the main difficulties is our poor comprehension of the internal processes that take

place in development teams. However, until such knowledge is available, empirical models that allow estimation may be constructed [2].

2 Introduction

An issue with current estimation models is that they depend on expert judgment and they require experienced personnel in order to be used [18]. Experienced personnel are scarce [18] and the current trend to agile methodologies with reduced, highly motivated and involved teams stresses the problem.

To improve estimation methods it is important to reduce the sources of variation in the development process as well as in the estimation process. Among the main sources of variation are: technology, expert judgment and size metrics that are not automatically obtained and not represent essential functional complexity, such as LOC and FP [15,22,23].

Making estimation independent of expert judgment would:

1. increase industry credibility
2. ease the relationship between client and developers by increasing its transparency
3. support joint change management
4. formalize the discussion among all involved stakeholders' deadlines and risk issues on an objective foundation [15]
5. contribute to evidence, on an objective basis, the risk introduced by fixing deadlines with non-technical criteria
6. might support contracting practices that are not based upon a price fixed at the point in which less is known about the project, but that instead allow to recomputed costs according to project execution environment, requirement volatility, organizational efficiency, average effort or development speed, system complexity etc.
7. supports small teams that lack experienced managers [16,22,23].

Disciplines such as architecture and civil engineering have elements such as mock-ups and blueprints, from which metrics may be extracted and scaled. This allows adjusting product design and, simultaneously, evaluating design impact on costs, deadlines and even the appropriate technologies for construction.

Even if they cannot reach the degree of formalization available in other disciplines, early indicators obtained from user data visions and models that -starting from those indicators- allow the estimation of project duration and costs would have an impact in the state of the art; they could prove useful in defining and managing software development projects. [15].

3 Conceptual Model

3.1 Early Assessment of the Essential Functional Complexity of an Information System

Assessing the functional complexity of an information system requires measures independent from technology and methodology used in the project, and independent from human expertise and other confounding factors. Such measures are useful only if they are available at very early stages in the development cycle, and based on the user information. Our hypothesis is that the system data set determines the complexity of the essential (non-redundant) functional requirements. Even if the final requirements could be expressed by various formats (queries, reports, etc.) by different groups of users, the essential functional requirements remain the same. The problem to solve is then to identify the data set used by the system.

At early stages, the system data set is uncertain. However, each user has a pretty good idea of his data subset. This information could be expressed in various forms (reports, queries, etc.) that constitute the external aspect of the system. The integration of all these fragmented visions of data will define the system data set. This task implies dealing with vision superposition, and redundancy. Nevertheless, the integration of all these visions contains the same non-redundant information [12].

From all these visions it is possible to measure the functional complexity of the data model by:

1. Integrating the data from user visions. This problem can be solved by using "synthesis of canonical visions" resulting in a relational scheme in third normal form [12].
2. Measuring the complexity of such integration by measuring the functional complexity of the data base schema.

3.2 Relational Data Base Complexity Metrics

For our purpose, the metrics proposed by Calero et al [5]: Number of Tables (NT); Degree of Referentiality, constituted by the number of foreign keys (RD), Number of Unique Attributes (NA) and Depth of Referential Tree (DRT) were most useful. These authors analyze in detail the formal properties of the proposed metrics, underlining the fact that they need an empirical support. From said metrics we developed an indicator of essential complexity for DIMIS that gives, within the limitations of our study, an empirical validation of the metrics.

These metrics are objective and automatically collectable. However, they are not independent from the relational database schema. Automating the design from the user

views solves this difficulty, and as a side effect the metrics could be collected in earlier stages. [20, 21, 22, 23, 24]. The influence of these metrics in the development time and effort has been studied in a set of estimation models [25].

3.3 Need for an Indicator

These complexity metrics for relational databases are not independent and have little user understanding. Instead of them, we use an indicator for essential complexity that represents their join effect and can be helpful in negotiating requirements. This indicator will be a function of the relational database complexity metrics presented above. The general form of the indicator is E-IC = f(NT, NA, RD, DRT).

4 Field Study

We observed 42 projects related to data-strong information systems [20]. From these post-mortem observations we selected 20 whose information was reliable and comparable. After developing empirical models for time and effort, two Essential Data Complexity Indicators (EDCIe and EDCIt) for Essential Data Intensive Management Information Systems (E-DIMIS) were defined. Both EDCI use the complexity metrics for relational data bases that were proposed by the ALARCOS group [5].
The complexity metrics for relational data bases were obtained automatically from the integration of the data views of final users. These metrics:

1. Are very early, if tools for automatic generation of relational data base schemas (representing the information contained in the set of user data views) are available.
2. Do not contain information related to detailed design, technological considerations or even the individual ways in which final users perceive their data.
3. Do not require expert judgment.
4. Are independent of the starting set of user data views.

The metrics properties are inherited by the indicators built on them. Therefore, the indicators are very early, formal and independent of expert judgment, design considerations, technology, the way in which final users perceive their data and other non-essential factors of the system.
The complexity indicators correlate well with the actual time and effort spent in the projects. The correlation is significant because it is supported by a number of causal analyses [19,20,21, 22, 23]. The indicators allowed the development of estimation models, and the models do not conflict with empirical observations, which are important properties [1,4, 5, 6,11,14].

4.1 Genexus

Genexus is a 4GL which generates the relational database and application programs required for the MIS under construction from its formal specifications. The tool stores information about the product specification in a knowledge database (KB). This data is very useful to capture metrics in an automated way.
The methodology behind the tool relies on two premises. First, in a middle or large company nobody has a global vision of the processes. Consequently, it is required to integrate different visions from various users. Second, requirements and database structures exhibit changes over the time.

The formal specification allows modeling the system, generating the database and code for the application.

4.2 The observed projects

We observed completed E-DIMIS (post mortem) developed by small groups of two to five people using a formal specification tool. The relational data base complexity metrics were obtained automatically from the specification. Project managers were asked to estimate the requirement volatility, which produced three sets of values: 30 %, 20% and 10%. Project managers also gave information about the project duration and total effort. Reported effort includes the whole staff.
Organizational efficiency was considered constant within the sample, because projects were developed by similar groups using a standard methodology supported by the development tool. Metrics extracted from the specification were obtained automatically.

5 The Essential Data Complexity Indicators for Time and Effort (EDCIe and EDCIt)

Salvetto [25] developed two formal models for very early effort and time estimation.
If we isolate from these models just the variables representing data structure complexity (DRT, RD, NA y NT) we may define two indicators:

$$EDCIE=DRT^{-2,234}\ RD^{-0,077}\ NA^{1,951}\ NT^{-0,516} \quad (1)$$

$$EDCIT=DRT^{-2,467}\ RD^{-0,209}\ NA^{1,84}\ NT^{-0,271} \quad (2)$$

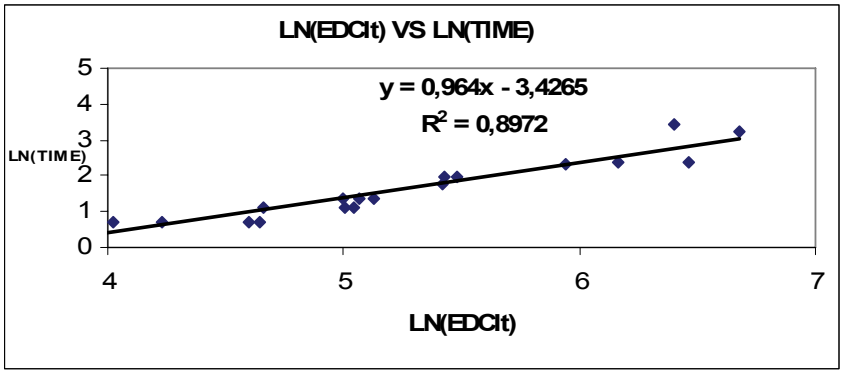


Fig. 1. LOG-LOG correlation graphic between Time and EDCIt

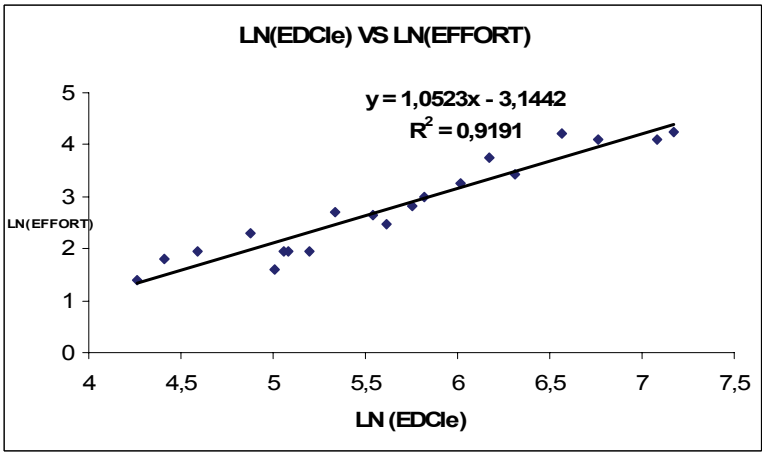


Fig. 2. LOG-LOG correlation graphic between Effort and EDCIe

6 Conclusions

Our results confirm agree with DeMarco´s [8] point that data-strong systems may be estimated from their data, and confirm the validity of the relational data base complexity metrics proposed by the ALARCOS group [5]. Our results also suggest that essential complexity for E-DIMIS depends strongly on the final users' data views. The complexity is made explicit by the indicators EDCIe y EDCIt.

This paper introduces two early indicators of complexity related to development effort and time. These indicators can be used in estimation models for E-DIMIS.

The use of such indicators can contribute to improve the communication between users and developers during requirements negotiation.

The size of the population studied limits the validity of our conclusions as often happens in empirical studies.

References

1. Boehm, B., Madachy, R. and Selby, R.: Cost Models for Future Software Life Cycle Processes: COCOMO 2.0. Ann. Software Eng. 1: 57-94 (1995)
2. Boehm, B., Sullivan, K.: Software Economics A Roadmap. University of Southern California University of Virginia Department of Computer Science Thornton Hall Los Angeles, CA 90089-0781 USA Department of Computer Science. http://www.cs.virginia.edu/~sullivan/publications/ICSE-2000-Roadmap.PDF
3. Briand, L., Khaled, L., Morasca, S.: Theoretical and Empirical Validation of Software Product Measures. This paper appears as Technical Report number ISERN-95-03, International Software Engineering Research Network, 1995.
4. Briand, L., Morasca, S. and Basili, V.: An Operational Process for Goal-Driven Definition of Measures. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 28, NO. 12, DECEMBER 2002 pp 1106-1125
5. Calero, C., Piattini, M., Polo, M., Ruiz, Grupo ALARCOS .Departamento de Informática,Universidad de Castilla La Mancha: Métricas para la evaluación de Complejidad de Bases de Datos Relacionales. Computación y SF.stemas Vol. 3, Nº 4, pp 264-273, 2000, CIC – IPN. ISSN 1405-5546.
6. Chulani, S., Boehm, B. and Steece, B.: Bayesian Analysis of Empirical Software Engineering Cost Models. IEEE Trans. Software Eng. 25(4): 573-583 (1999)
7. Construx Software Inc. http://www.construx.com/estimate.
8. DeMarco, T.: Controling Software Projects. Yourdon Press, New York, 1982.
9. DEPARTMENT OF THE AIR FORCE, Software Technology Support Center Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems Version 4.1, Condensed Handbook , 2003.

10. DEPARTMENT OF THE AIR FORCE, Software Technology Support Center. Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems, Command and Control Systems, Management Information Systems Version 3.0, Fall 1999.
11. Fenton, N. and Pfleeger, S.: Software Metrics. A Rigorous & Practical Approach. PWS Publishing Co. 1997.
12. Gonda, B.: ¿Desarrollo orientado a procesos u orientado a datos? Algunas reflexiones en el 40° aniversario de los Sistemas de Gerencia de Bases de Datos, ARTech, 2003 www.genexus.com/whitepapers
13. Humphrey, W.S.: Your Date or Mine. In The Watts New Collection. Software Engineering Institute, Carnegie Mellon University, http://interactive.sei.cmu.edu/, 2001.
14. Juristo, N. & Moreno, M.: Basics of Software Engineering Experimentation. Kluwer Academic Publishers 2001.
15. Kavoussanakis, K., Sloan T.: THE UNIVERSITY OF EDINBURGH UKHEC Report on Software Estimation Document EPCC-UKHEC 0.1
16. Latorres,E, Salvetto, P., Nogueira, J.,Larreborges,U.: Una herramienta de apoyo a la gestión del proceso de desarrollo de software. Proceedings CACIC 2003. La Plata, 2003.
17. McConnell, S. :Rapid development: taming wild software schedules. Microsoft Press, 1996.
18. Nogueira, J.: A Formal Model for Risk Assessment in Software Projects. PhD Dissertation. Naval Postgraduate School, Monterrey, California.September,2000.
19. Salvetto, P. and Nogueira, J. Size Estimation for Management Information Systems Based on Early Metrics :An Automatic Metric Tool Based in Formal Specifications. Proceedings of the International Conference on Computer Sience, Software Engineering,Information Technology, e-Business and Applications (CSITeA'03), june 5-7, 2003 Rio de Janeiro, Brazil in Cooperation with the International Society for Computers and Their Applications (ISCA), USA Winona State University (WSU), USA Universidad Nacional de San Luis (UNSL), Argentina Net of National Universities with Computer Science Careers (RedUNCI), Argentina. Pags 72-77.ISBN 0-9742059-0-7.
20. Salvetto, P., Carrillo, J., Marbán, Ó, Fernández, J., Nogueira, J. y Segovia, J.: Indicadores Empíricos Formales y muy Tempranos de Complejidad Esencial de Sistemas de Gestión Intensiva de Datos: un modelo conceptual. II Conferencia Internacional Encuentro ISBSG-AEMES VI Conferencia Anual de la Asociación Española de Métricas de Sistemas Informáticos. Métricas TI: La experiencia en la Mejora de Procesos TI. 4-5 de octubre de 2005, Madrid, España. (Publicado en la Revista de Procesos y Métricas de la Asociación Española de Métricas de Software Volumen 3, Número 1, Abril de 2006, ISSN 1698-2029).
21. Salvetto, P., Martínez, M., Luna, C. and Segovia, J.: A Very Early Estimation of Software Development Time and Effort Using Neural Networks. X CONGRESO ARGENTINO DE CIENCIAS DE LA COMPUTACIÓN 2004 (CACIC04) San Justo, Buenos Aires. Argentina, octubre de 2004.
22. Salvetto, P., Nogueira, J y Segovia, J: Gestión de Cambios Apoyada por Modelos Formales de Estimación de Tiempo y Esfuerzo. 4 tas JORNADAS

IBEROAMERICANAS DE INGENIERÍA DE SOFTWARE E INGENIERÍA DEL CONOCIMIENTO JIISIC'04. Facultad de Informática Universidad Politécnica de Madrid. 3-5 de noviembre de 2004. Madrid, España

23. Salvetto, P., Nogueira, J y Segovia, J: Modelos Automatizables de Estimación muy Temprana del Tiempo y Esfuerzo de Desarrollo de Software de Gestión .XXX CONFERENCIA LATINOAMERICANA DE CIENCIAS DE LACOMPUTACIÓN (CLEI 2004) 27 de septiembre – 1 de octubre 2004. Arequipa Perú.
24. Salvetto, P., Nogueira, J., Fernández, J y Segovia, J.: Una Verificación Empírica deModelos Automatizables de Estimación muy Temprana de Proyectos de Desarrollo de Sistemas de Gestión. 4 tas JORNADAS IBEROAMERICANAS DE INGENIERÍA DE SOFTWARE E INGENIERÍA DEL CONOCIMIENTO JIISIC'04. Madrid, España.
25. Salvetto, P.: Modelos Automatizables de Estimación muy Temprana del Tiempo y Esfuerzo de Desarrollo de Sistemas de Información. Tesis Doctoral. Facultad de Informática Universidad Politécnica de Madrid. Madrid, España. 29/11/2006.
26. Sommerville, I.: Software Engineering, Sixth Edition. Addison-Wesley Publishers Limited, 2001.

Annex 1 Definitions, Axioms and Assumptions on which the research is based.

Definition 1: A Data Intensive Management Information System (DIMIS) is a management information system which:

1. Does not process object oriented databases, geographic databases, complex object and other data features uncommon in traditional MIS.
2. Where high complexity algorithms are rare or inexistent.
3. Does not have critical time requirements.

Definition 2: Attribute. Atomic data, with semantics defined for some DIMIS user, to which a type and a unique name are associated. If two attributes has the same name then their semantic is the same.

Definition 3: User data view (UDV) A set of attributes that represent one user's vision of an entity to be represented in the DIMIS, such as the user perceives it, without having any design, normalization, or other criteria applied to it by system developers.

Definition 4: System user data views (SUDV) Set of data user views (UDV), not necessarily disjoint, of a DIMIS, in which attributes semantically identical have the same name.

Axiom 1: Need for integration of User Data views (UDVs) In a medium or large organization, nobody has a global view of data and processes. Therefore, there is a need to integrate the individual UDVs in system in order to construct an integral view of the data.

Definition 5: Essential DIMIS (E-DIMIS) is a DIMIS in which only non-redundant functionality was developed, on the basis of the SUDV, integrated from user views and requirements.

Definition 6: Global Empirical Model of Formal Estimation (GEMFE). Empirical model for global estimation that allows the estimation of the total development effort or time for an E-DIMIS, does not require expert judgment, and may be automated.

Definition 7: Essential Complexity Indicator of an E-DIMIS (E-CI). A real number that may be obtained before any design activity takes place and that does not require expert judgment, starting from any SUDV of the E-DIMIS, and that is used as an independent variable of a GEMFE for the E-DIMIS for which it was obtained.

Axiom 2: The E-CI is independent of the SUDV from which it was obtained. This is intuitively justified because the E-DIMIS under study are based on data, with a minimal proportion of high complexity algorithms, and because user views are not influenced by the technology, design concepts or other criteria alien to the system itself, introduced by designers. Although the same system could be perceived by users in multiple ways, the resulting SUDV must contain the information needed to develop it. Therefore, from that point of view, the different user perceptions are equivalent.

Axiom 3: Continuous Change inherent to information systems. Requirements, environment and user data views are subject to constant and unavoidable changes with time. Therefore, the integration of the UDV in a SUDV must be automatic and continuing.
Definition 8: Essential information of a SUDV (EIV). The maximum quantity of non-redundant information on the attributes of some entity or the relationships that may be inferred among entities from the SUDV in which they are featured.
Definition 9: Essential Functionality of a SUDV (EFV). The maximum non-redundant functionality that may be obtained from its EIV.
Axiom 4: The Essential Information of all the SUDV of an E-DIMIS is the same.
Axiom 5: A single SUDV (any) of an E-DIMIS determines the E-CI of the E-DIMIS.
Definition 10: Potential Functionality of an E-DIMIS. The portion of the EFV that has not yet been developed.
Axiom 6: The E-CI is an indicator of the EFC and its semantics is the "potential information of the data structure of the information system". The E-CI could be described as the potential information of the system data structure. It shows the maximum information and functionality that could be achieved, given the system data structure. If more functions are added, new data and/or new relationships would be needed, increasing complexity or introducing redundant functionality, which would contradict definition 5.
Assumption 1: The final Potential Functionality of an E-DIMIS will be null. The EFV is the most that can and will be done, since it was obtained from the users' data vision, and if users could once evoke a piece of data in their memory, some day they may evolve a requirement associated with that piece of data. So, at the end of the development process, it will not be possible to add new, non-redundant functionalities that are useful to the business or to some user without including new data or new relationships among existing data. In other words, the "potential information" of the data will have been fully converted into information and non-redundant functionality.
Axiom 7: An E-CI contains useful information as an input to a GEMFE. The final product of the transformation of the "potential information" in the data structure is not determined. Different users may request the functionality and information that may be obtained from the data structure in different ways, but the effort needed to produce it will be, under similar conditions, the same.
Assumption 2: Reduction of variability has an impact in the global estimation of results. Even though we do not understand in depth the complex internal processes and interactions that take place during the development process, we can observe the process and build models that estimate well its global results, if we reduce the sources of variability by using tools for automatic data base design, automatic code generation and module integration, and that support the work of a small group of developers, integrating users and following a standard methodology.
Assumption 3. It is possible to empirically determine E-CI from the observation of the role played by data complexity metrics in GEMFEs.
Definition 11: Formal DIMIS Specification (FSS). A specification of an information system that –given an adequate generator- has enough elements to automatically create

the code needed to operate the system in a wide set of programming languages and execution platforms, starting from a FDS (see next definition).

Definition 12: Formal data specification of a DIMIS (FDS). Specifications of some SUDV that contains enough information about the data and relationships that the system must handle, in order to automatically infer a relational data base schema that represents the information in the SUDV.

Definition 13: Formal DIMIS Data Specification Tool of a DIMIS (FDST). A software tool that supports the activities of development and storage of a FDS.

Definition 14: Data Structure Generation Tool A software tool that, starting from a FDS, generates a relational data base schema that represents the FDS.

Definition 15: Formal DIMIS Specification and Development Tool (FDSDT). A software tool that supports the activities of maintenance and development of a FSS, with at least the following capacities:

1. To specify the SUDV
2. To treat attributes with the same name as the same, independent of where the attributes are referenced, and to infer, starting from the attributes, the relationships among the entities represented in the DIMIS.
3. To generate a relational data base schema that represents the information contained in the SUDV.
4. To maintain the UDVs that constitute the SUDV of the DIMIS, allowing
   - To eliminate a UDV
   - To create a new UDV
   - To eliminate an attribute of an UDV
   - To add an attribute to a UDV
5. To specify relationships among UDVs
6. To specify processes by means of references to data, based on the SUDV, independently of the relational data base schema.
7. To automatically generate a relational data base schema, representing the information contained in the SUDV.
8. To automatically generate code for the system processes in different languages and execution platforms
9. To automatically evaluate the impact of a change in the SUDV on the relational data base schema, to report it, and, if the change is confirmed and consistent, then
10. to generate automatically the new relational data base schema representing the new SUDV
11. to automatically generate the code needed to migrate data from the old schema to the new
12. to automatically generate code for those system processes that were affected by the change
13. to automatically integrate, as long as there are no repeated attribute names, the specifications of different DIMIS to produce a single one, representing the integrated view of the individual systems, reporting inconsistencies if needed

Requirements to Components: A Model-View-Controller Architecture

Sabnam Sengupta
Department of Computer Science & Engineering,
Jadavpur University, Kolkata – 700032, India.
sabnam_sg@yahoo.com

Abhik Sengupta
Cognizant Technologies Solutions
Salt Lake, Sector V,
Kolkata - 700091, India.
Abhik.sengupta@cognizant.com

Abstract: In this paper, we propose a framework where functional requirements are traced from use case model to component model via analysis and design models. Here, components of the component models are derived by grouping and packaging design classes based on the type of analysis classes they are derived from. As there are three different types of analysis classes: boundary, controller and entity, the design classes derived from the corresponding analysis classes get classified at the first iteration. The components thus derived using this approach form the components of Model View Controller Architecture; different components having design classes of similar functionalities. This framework can be used to verify and ensure that use case flow of events is traced in analysis model and then to component model via design models. The architecture with the components designed using this framework also ensures separation of concerns, roles among the components to achieve high cohesion and low coupling.

Keywords: Model-View-Controller Architecture, Use case model, analysis model, design model, component model, Component based architecture, UML Component diagram, XML.

1. Introduction

Component-Based development, realizing the intuitive and attractive idea of rapidly obtaining complex software systems by the assembly of simpler components, has long captivated the industrial practitioners with the promise of cheaper products with higher reliability and maintainability. A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Reusability, whose benefits include both the reduction of costs and time-to-market of software products, is a key issue in software engineering. Component-based software development has emerged to increase the reusability and interoperability of pieces of software. Component-based development aims at constructing software artifacts by assembling prefabricated, configurable and independently evolving building blocks, the so-called components.

However, it is only via a rigorous design discipline and by adopting standard modeling notations as well as strict documentation and design rules that components independently built can effectively interact. This is the basic notion of Design-by-Contract [3], discipline originally conceived for Object-Oriented systems, but even better suited for Component Based development; indeed Objects and Components, though differing concepts, share many aspects.

In recent years, the focus of software development has progressively shifted upward, in the direction of the abstract level of architecture specification. High-level and standardized models must be adopted, in such a way that the consistency (compatibility and the interoperability) among components can be verified as early as possible. The widespread adoption of the Unified Modeling Language (UML) evidences this trend, and

its flexibility to specialized yet standard-compatible extensions, where necessary, provides a valuable tool for pursuing this direction. In particular, Cheesman and Daniels [1] describe how UML can be specialized for modeling within a Component Based paradigm embracing the basic principles of the Design-by-Contract approach [3]. Very recently, the OMG Model Driven Architecture (MDA) approach to development pursues a complete separation between the base platform-independent model of an application, and the descriptions of one or more platform-specific models, describing how the base is implemented on each of the supported platforms [2]. The base model in MDA is specified in UML.

Model-view-controller (MVC) is an architectural pattern used in software engineering. In complex computer applications that present lots of data to the user, one often wishes to separate data (model) and user interface (view) concerns, so that changes to the user interface do not affect the data handling, and that the data can be reorganized without changing the user interface. The model-view-controller solves this problem by decoupling data access and business logic from data presentation and user interaction, by introducing an intermediate component: the controller.

This work focuses in that direction of designing and packaging components with the design classes. These design classes trace the analysis model that is derived from the use case model. As there can be three types of analysis classes: boundary, controller and entity, the design classes also get categorized accordingly. The components, too, being derived by packaging design classes of similar functionalities get their roles defined at a very early stage. There can be three types of roles:

Model: Domain specific representation of the information, i.e., the data models,

View: Components, through which the users interact with the system,

Controller: Components that get invoked directly with the user interaction and they invoke the appropriate models based on the user input.

These components build Model-View-Controller (MVC) architecture, which is essentially component-based architecture.

2. Review of Related Works

Lots of research works are going on in the field of Component based architecture. In most of these works, special interest has been given recently to the reconfiguration and migration of components in component-based system. Some design methodologies addressing component-based development have been proposed recently. Most of them are based on the UML [7], c.f. [4, 5, 6].

Cheesman and Daniels [1] describe how UML can be specialized for modeling within a Component Based paradigm embracing the basic principles of the Design-by-Contract approach [8]. Amber [8] is an abstract component based modeling language combines a model-based approach [8] with a UML-based approach [5]. This combined approach aims at profiting from the advantages of both approaches. Catalysis [4] is another complex software development process based on UML. Similarly to the Unified Process [5], Catalysis is much like a process template, which can be tailored according to a particular development project. Catalysis being flexible and scalable, it is popular among software developers. A major benefit of Catalysis is its explicit use of components.

However, being a broad software development process, Catalysis is not completely component-oriented.

As in recent time the focus in the software industry has moved into reusability of components and creation of repository of components, it would be effective if we can design a component model where the roles of different components are very well defined that would make the components easily replaceable and modifiable as the components become highly cohesive and less coupled with each other. With that aim we propose a Model-View-Controller architecture in designing a component that trace functional requirements from use case model to analysis model to design model and then to component models.

3. Scope of Work

A software component is a physical, replaceable part of a software system that packages implementation and provides realization of a set of interfaces. When we are considering software architecture based on components, a component should have a specification, it should have an implementation, it should conform to some standards, it has to be package able into modules, and it should be deployable. A component specification is usually complete; it contains all the information that a client of the component needs to know. A component specification makes it easier to buy, sell, and replace components -- if the component fulfils its contract, it should function correctly in the system. From a client's perspective or user's perspective, there is no need to explore into a lower level of detail before using the component.

But, from an architectural perspective it is extremely important how these components are built so that there is a very clear separation of concerns. The architecture based on these components can achieve a very high cohesion and very low coupling. With that vision, we here propose to derive component models that trace to the use case models that handle functional requirements specified in the Software Requirement Specification (SRS) Document. Analysis classes in the analysis model realize use case models and analysis classes are traced to design classes.

As analysis classes are classified as: boundary, control and entity classes, the design classes also get classified accordingly.

Packaging design classes that play similar kind of roles in the architecture derives the components of component model. This approach is illustrated in Figure 1.

When different components are assembled, they are assembled through the interfaces they implement. Interfaces are specifications of services provided by classes or components.

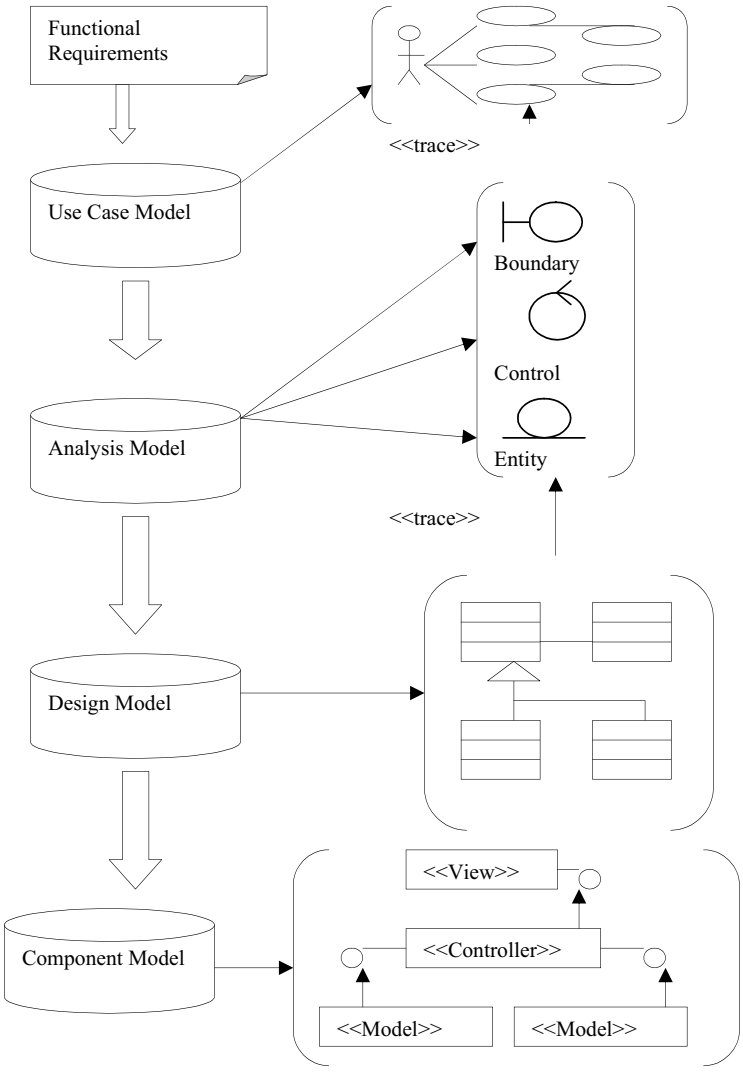


Figure 1: From Use Case Model to Component Model: A Model-View-Controller Framework

Interfaces are most closely associated with components; a component without an interface may be technically well formed, but suspect. Functional Specification of these components and consistency verification among them are very important to ensure cost effectiveness at a very early stage of deployment.

In this paper, we propose a component based architectural framework that follows Model-View-Controller design pattern where roles of the components are well defined; ensuring separation of concerns. Packaging design classes of similar roles; ensuring high cohesion and low coupling build these components. We propose several XML schemas for different models used in different phases of software development.

4. Functional Requirements to Components

Traditional object-oriented software development aims at providing reusability of object type definitions (classes), at design and implementation levels. In contrast, component based development aims at providing reusability of components at deployment level. In this way, components represent pieces of functionality that are ready to be installed and executed in multiple environments. In this paper, we propose to address a design issue in component-based development, i.e., separation of concerns, roles of components to achieve low coupling and high cohesion among the components. For that, we propose some restrictions, following best practices that are adhered to by most designers in any case, in designing these components. The restrictions are:

1. Each use case of use case model has to be traced in
    a. One or more boundary classes
    b. One or more entity classes
    c. One or more control classes

In the analysis model

2. An analysis class has to be traced in one or more design classes.
3. If a design class in design model traces more than one analysis classes, then the type of the analysis classes has to be the same, i.e., either one of boundary, control and entity.
4. When a design class or a set of classes are grouped and packaged to build components, the design classes that trace same type of activity classes have to be grouped. For example, if there are three design classes, all of them tracing "boundary" type analysis classes, they can build a component. But, if they trace different types of design classes they may not be grouped to build a component.

The restriction 4 ensures that each component plays a unique role in the architecture.

a. They can act as "interfaces" through which the users interact with the system, or, (View)
b. They can act as components that invoke different data models based on user input, or, (Controller)
c. They can act as the data models themselves. (Model)

This builds MVC architecture.
In the following sections, we propose XML schemas for different models used in software development. These XML schemas conform to OMG's XMI standard. Java programs that use XML parser are used to verify the restrictions proposed.

4.1 Use Case Models to Analysis Models

UML use case diagrams have become the de-facto standard for defining and capturing functional requirements. In Unified software development process use case models consist of UML use case diagrams. A use case diagram is composed of use cases, their actors, their relationships and their flow of events (also known as Activity Flows). In the analysis model, a use case is realized in collaboration and collaborations are mapped to the analysis classes. There can be three types of analysis classes: Boundary, Control and Entity. Boundary classes in general are used to model interaction between the system and its actors. Entity classes in general are used to model information that is long-lived and often persistent. Control classes are generally used to represent coordination, sequencing, transactions, and control of other objects. And it is often used to encapsulate control related to a specific use cases. Here, in order to trace functional requirements into component design, we propose a restriction: A use case has to be traced in
This translation is diagrammatically represented in Figure 2.

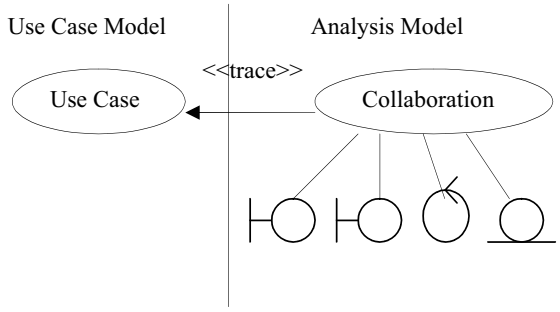


Figure 2: Requirement tracing from use case models to analysis models

For this tracing, we propose XML schemas representing use case model and analysis model as shown in Figure 3 and 4 respectively.

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsd=http://www.w3.org/2001/XMLSchema xmlns:p="URI">
<xsd:import namespace=http://www.omg.org/XMI
schemaLocation="xmi20.xsd"/>
<xsd:complexType name="UseCaseModel">
<xsd:sequence>
   <xsd:complexType name="UseCase">
      <xsd:element name="ucId" type="xsd:integer"/>
      <xsd:element name="ucName" type="xsd:string"/>
      <xsd:complexType name=”Events”>
            <xsd:complexType name="Event">
            <xsd:attribute name="eventId" type=”xsd:integer”
      use=”required”/>
            <xsd:element name="eventDesc" type=”xsd:string”/>
            <xsd:complexType name=”tracedByAnalysisClasses” >
            <xsd:element name=”analysisClass” type=”xsd:string”
      />
      </xsd:complexType>
      </xsd:complexType>
   </xsd:complexType>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Figure 3: XML Schema representing Use Case Models

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsd=http://www.w3.org/2001/XMLSchema xmlns:p="URI">
<xsd:import namespace=http://www.omg.org/XMI
schemaLocation="xmi20.xsd"/>
<xsd:complexType name="AnalysisModel">
<xsd:sequence>
      <xsd:complexType name="AnalysisClass">
            <xsd:attribute name="type" type="xsd:string"/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:complexType name=”EventsDealt”>
            <xsd:sequence>
                  <xsd:complexType name="Event">
                  <xsd:attribute name="id" type=”xsd:integer”
            use=”required”/>
            </xsd:sequence>
            </xsd:complexType>
      </xsd:complexType>
</xsd:sequence>
</xsd:complexType>
```

```
</xsd:schema>
```

Figure 4: XML Schema representing Analysis Models

These XML schemas follow the XMI standards.

A Java program that uses an XML parser is used to verify if all the events in activity flows of use case model has got mapped to the analysis classes in the Analysis model. The same program is also used to ensure that the type of analysis classes can be either one of boundary, control and entity.

4.2 Analysis Models to Design Models

The analysis classes derived in the analysis models get traced to the design classes of the Design Model. This is shown in Figure 5.

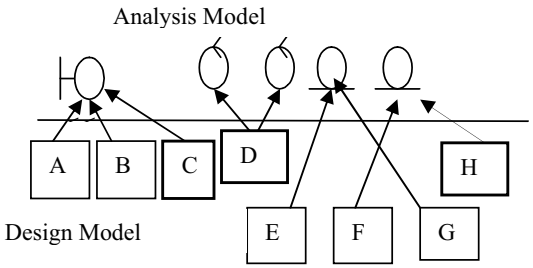


Figure 5: From Analysis Models to Design Models

Here we propose a XML schema representing a design model in Figure6.

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsd=http://www.w3.org/2001/XMLSchema xmlns:p="URI">
<xsd:import namespace=http://www.omg.org/XMI
schemaLocation="xmi20.xsd"/>
<xsd:complexType name="DesignModel">
<xsd:sequence>
       <xsd:complexType name="DesignClass">
              <xsd:element name="name" type="xsd:string"/>
              <xsd:complexType name="TracedAnalysisClasses">

              <xsd:element name=”analysisClass” type=”xsd:string”/>
       </xsd:complexType>
```

```
      </xsd:complexType>
</xsd:sequence>
</xsd:complexType></xsd:schema>
```

Figure 6: XML Schema representing a Design Model

Similarly a Java program that uses an XML parser is used to verify that each analysis class in Analysis model gets traced to one or more design classes. The same Java program is also used to ensure that if a design class in design model traces more than one analysis classes, then the type of the analysis classes has to be the same, i.e., either one of boundary, control and entity.

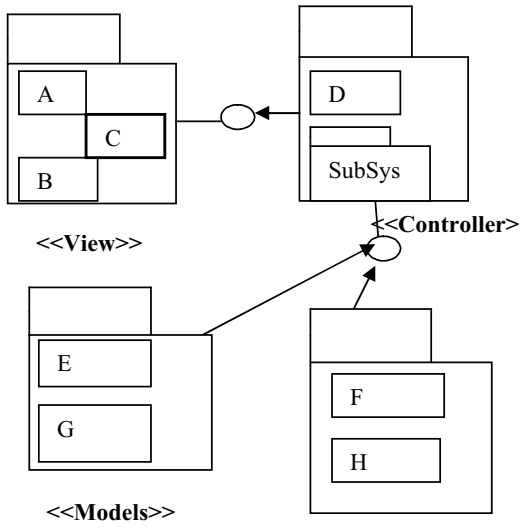


Figure 7: Component Model

4.3 Design Models to Component Models

The Design classes that trace the activity classes are packaged to build components. This is shown in Figure 7.

Here, we propose a XML schema that represents the component model as shown in Figure 8.

```
<xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="URI"
xmlns:xmi="http://www.omg.org/XMI"
```

```
xmlns:xsd=http://www.w3.org/2001/XMLSchema xmlns:p="URI">

<xsd:import                          namespace=http://www.omg.org/XMI
schemaLocation="xmi20.xsd"/>
<xsd:complexType name="ComponentModel">
<xsd:sequence>
       <xsd:complexType name="Component">
              <xsd:attribute      name="id"      type="xsd:integer"
              use=”required”/>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="type" type="xsd:string"/>
              <xsd:complexType name=”DesignClasses”>
                     <xsd:complexType                  name="class">
                  <xsd:attribute    name=”name”    type=”xsd:string”
              use=”required”>
              </xsd:complexType>
       </xsd:complexType>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Figure 8: XML Schema representing a Component Model

A Java program that uses a XML parser is used to verify that the design classes that build a component trace the same type of analysis class in the analysis model; as stated in restriction 4.

In the following section we explain our approach with the help of a case study.

5. Case Study

We have considered a simple example of a Library System where a member can register, cancel membership, issue and return books from the library. The use case diagram is shown as in Figure 9.

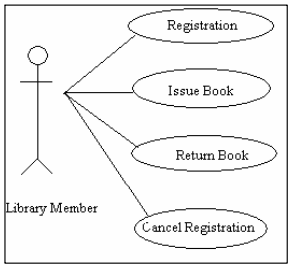


Figure 9: Use Case Model of Library System

The Flow of events for “Registration” is:

1. Person details are entered.
2. Checking is made whether an existing member or not.
3. If not an existing member, membership is created and a member ID is generated.

The Flow of events for “Issue Book” is:

1. Member ID is entered and validated
2. Every Member has a maximum allowable limit for issuing books, which depends on member category. Check whether member is allowed for issuing books
3. If issue is allowed accept Book ID and validate it.
4. Check if the book is already issued to the member and needs re-issue.
    a. If re-issue request then check if there is any demand pending
    b. If yes, re-issue request rejected.
    c. If no, the book is re-issued.
5. If request is for issue
    a. Check for availability of book
    b. If available, issue the book
6. Otherwise, place demand on hold.

The Flow of events for “Return Book” is:

1. Member ID is entered and validated.
2. Book ID is entered and validated.
3. Checking is made whether that book was issued to that member or not.
4. Book data is updated.
5. Member data is updated.

The flow of events for “Cancel Membership” is

1. Member ID is entered and validated
2. Check if any book is already issued to the member or not.
3. If no book is issued to the member, the membership is cancelled.

The part of the XML file representing this use case model is shown in Figure 10.

```
<?xml version="1.0"?>
<UseCaseModel>
   <UseCase>
```

```
<ucId> 01 <ucId>
<ucName> Register </ucName>
  <Events>
    <Event eventId = '1.1'>
      <eventDesc>
          Person details are entered.
      </eventDesc>
      <tracedByAnalysisClasses>
      <analysisClass>
          Authorization Interface
      </analysisClass>
      </tracedByAnalysisClasses>
    </Event>
    <Event EventId = '1.2'>
      <eventDesc>      Checking is made whether an existing
    member or not.
      </eventDesc>
      <tracedByAnalysisClasses>
      <analysisClass>  Authentication
      </analysisClass>
      <analysisClass>
      Member
      </analysisClass>
      </tracedByAnalysisClasses>
    </Event>
          ……
      </Events>
  </UseCase>
<UseCase>
    <ucId> 01 <ucId>
    <ucName> Issue Book </ucName>
      <Events>
          <Event EventId = '2.1'>
            <eventDesc></eventDesc>
            <tracedByAnalysisClass>
            </tracedByAnalysisClass>
          </Event>
          <Event EventId = '2.2'>
            <eventDesc></eventDesc>
            <tracedByAnalysisClass>
            </tracedByAnalysisClass>
          </Event>
          ……
      </Events>
  </UseCase>
  ……
</UseCaseModel>
```

Figure 10: XML Document representing Use Case Model of Library System

The use cases of the Use Case Model are traces to the Analysis classes that build the analysis model as shown in Figure 11.

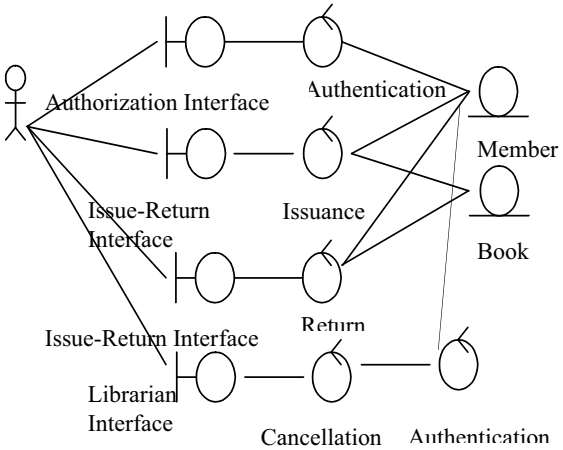


Figure 11: Analysis Model of a Library System

The part of the XML file that represents the analysis model is shown in Figure 12.

```
<?xml version="1.0"?>
<AnalysisModel>
  <AnalysisClass type='boundary'>
    <name> Authorization Interface </name>
    <EventsDealt>
       <Event id='1.1' />
    </EventsDealt>
  </AnalysisClass>
  <AnalysisClass type='control'>
    <name> Authentication </name>
    <EventsDealt>
       <Event id='1.1' />
       <Event id='4.3 />
    </EventsDealt>
  </AnalysisClass>
  ……..
</AnalysisModel>
```

Figure 12: XML Document representing Analysis Model of a Library System

The Analysis classes are traced to the design classes. The process is depicted in Figure 13 of Appendix.

From this, we can derive the design model of the Library System as shown in Figure 14 of Appendix. The part of the design model is represented in XML as shown in Figure 15.

```
<?xml version="1.0"?>
<DesignModel>
       <DesignClass>
          <name>Display</name>
          <TracedAnalysisClasses>
             <analysisClass> Authorization-Interface
             </analysisClass>
             <analysisClass> Issue-Return-Interface
             </analysisClass>
          </ TracedAnalysisClasses >
       </DesignClass>
       ……..
       <DesignClass>
          <name>Transaction-Management</name>
          <TracedAnalysisClasses>
             <analysisClass>
             Issuance
             </analysisClass>
             <analysisClass>
             Return
             </analysisClass>
          </ TracedAnalysisClasses>
       </DesignClass>
</DesignModel>
```

Figure 15: XML Document representing Design Model of a Library System

We now propose to package classes into components to generate the Component Model of the Library System as shown in Figure 16.

It is a Component-Based Model-View-Controller Architecture. Part of a Component Model of the Library System is represented in XML format in Figure 17.

```
<?xml version="1.0"?>
<ComponentModel>
       <Component id='01'>
          <name>Library-Interface </name>
          <type>View</type>
          <DesignClasses>
                 <class name="Display"/>
                 <class name="Keyboard"/>
                 <class name="Librarian"/>
          </DesignClasses>
       </Component>
       <Component id='04'>
          <name>Transaction-Management</name>
```

```
        <type> Controller   </type>
        <DesignClasses>
      <class name="Transaction-Manager"/>
   </DesignClasses>
   </Component>
</ComponentModel>
```

Figure 17: XML Document representing Component Model of a Library System

6. Conclusion

In this paper, we propose a framework for designing a system based on Component Based Architecture. Here, we propose to achieve separation of concerns among these components using Model-View-Controller design pattern. This would enable us building components that are highly cohesive and less coupled with other components making them easily replaceable and modifiable. Reusability of such components will be high as the roles of the components are going to be well defined and non-overlapping.

References

[1] Cheesman J., Daniels J., UML Components, A simple process for specifying component-based software, Addison Wesley, 2001

[2] Model Driven Architecture, A Technical Perspective.' doc. n. ab/2001-01-01, n. ormsc/2001-07-01

[3] Meyer B. Applying 'Design by Contract'. Computer, 25 (10), Oct. 1992, 40-52

[4] D'Souza, D. F. and Wills, A. C.: Objects, Components and Frameworks with UML: the Catalysis Approach. Addison Wesley, USA, 1999.

[5] de Farias, C.R.G., Ferreira Pires, L. and van Sinderen, M.: A component-based groupware development methodology. In Proceedings of the 4th Int. Enterprise Distributed Object,Computing Conference (EDOC'00), pp. 204-213, 2000.

[6] Jacobson, I., Booch, G. and Rumbaugh, J. The unified software development process. Addison Wesley, USA, 1999.

[7] Object Management Group UML Revision Task Force: OMG UML v. 1.3: Revisions and Recommendations, 1999.

[8] Quartel, D.A.C., van Sinderen, M.J., and Ferreira Pires, L.: A model-based approach to service creation. In Proceedings of the 7th International Workshop of Future Trends in Distributed Computing (FTDCS'99), pp. 102-110, 1999.

Appendix

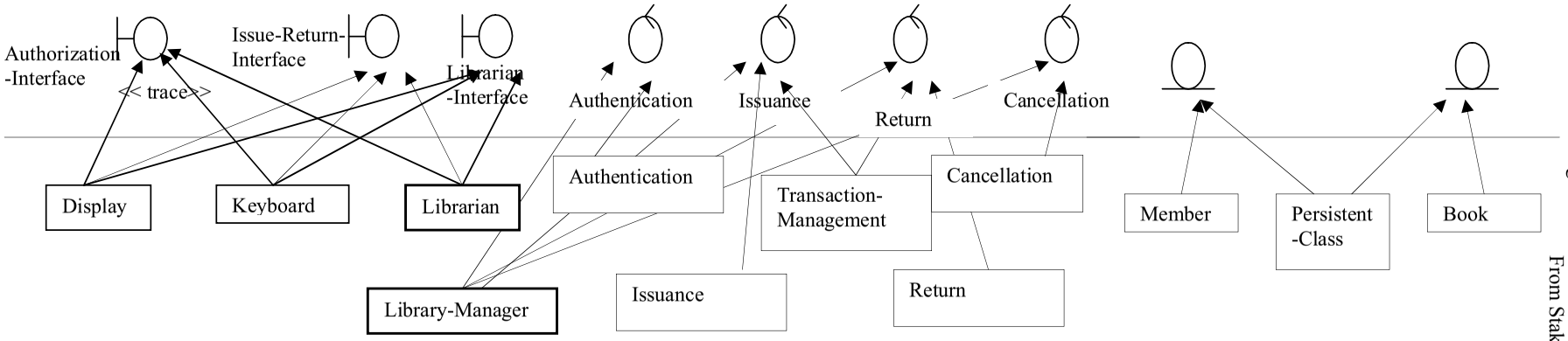


Figure 13: Analysis Classes Traced to Design Classes

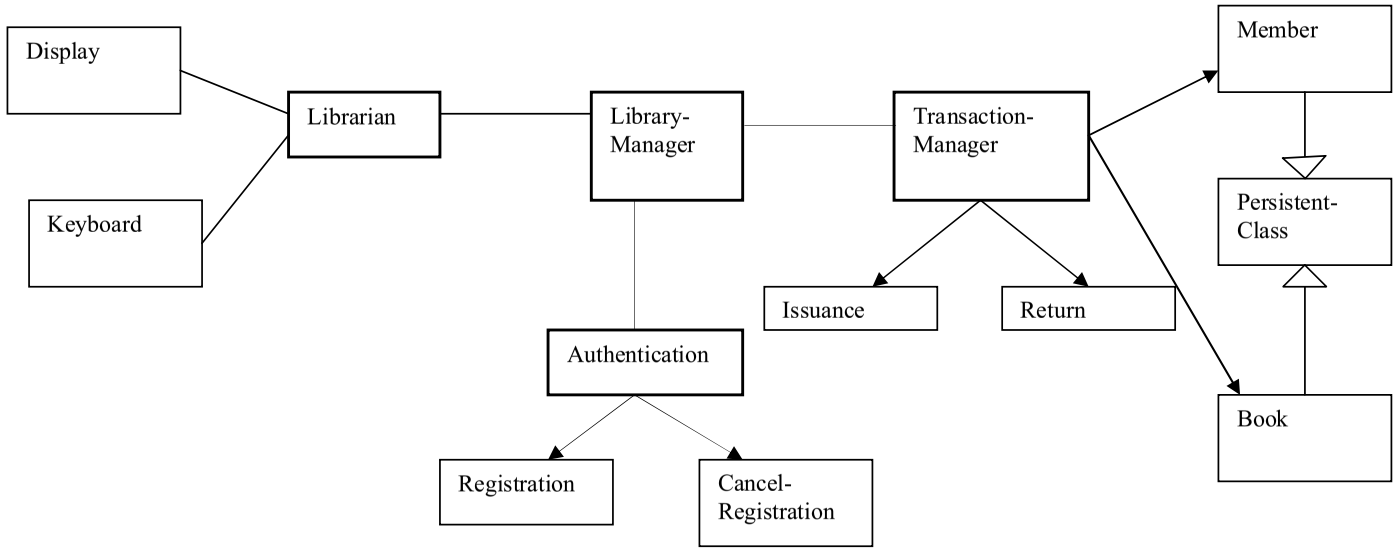


Figure 14: Design Model of Library System

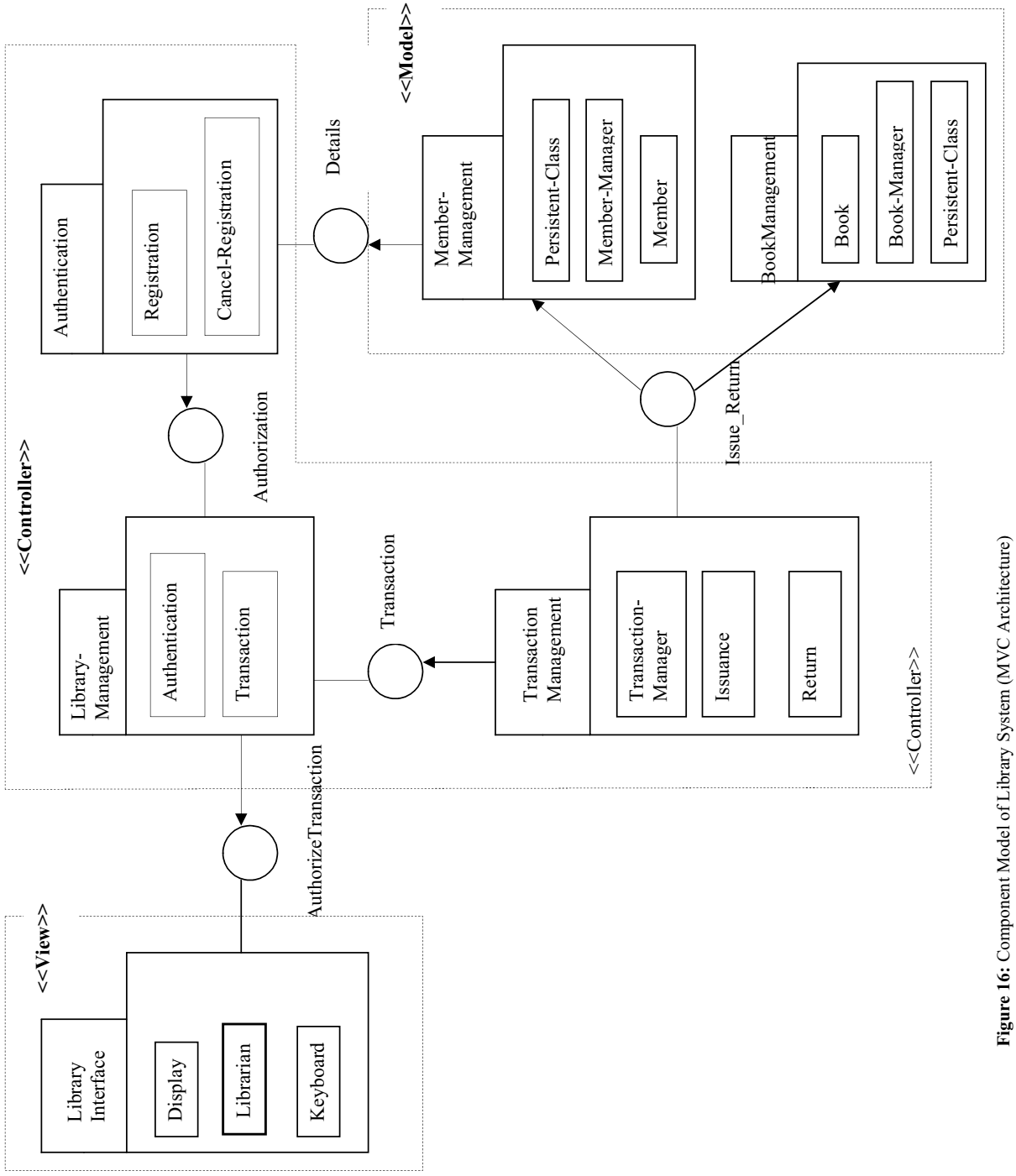


Figure 16: Component Model of Library System (MVC Architecture)

Composition and Reconciliation: Challenges and Possible Solutions to Integrate Stakeholders' Needs Together?

Stephen S. Yau and Zhaoji Chen
School of Computing and Informatics
Arizona State University
Tempe, AZ 85287-8809 USA
Email: {yau, zhaoji.chen}@asu.edu

Today people increasingly rely on information systems, which often consist of software systems running on various interconnected computing devices, for almost every aspect in our lives. As stakeholders' needs increase, systems become more error-prone. Misunderstanding of various stakeholders' needs, miscommunication among different developers, and oversight of certain requirements can all be the causes for losses.

New techniques are being developed to support effective and collaborative software development for large-scale systems through composition of smaller, more manageable units. For example, the emergence of service-based architecture offers developers new opportunities to rapidly develop large-scale distributed information systems by composing massively available services, which are independently developed, regardless of the programming languages and platforms used to develop and run these services. These approaches certainly help improve the quality of large-scale information systems because the component units are easier to develop and test, and composition and reuse of well tested components can greatly reduce developers' workload. However, without proper support, the composition approach may cause new problems while solving some of the old ones. Even with simple robust components, when they need to collaboratively provide certain functionality, the challenge is how we can address new playground for shared objects while maintain their robustness. It is much more difficult to make all stakeholders happy at the same time. There are three major challenges.

First, to express their needs, different stakeholders may use their own vocabularies, which are not standardized. For example, auditability requirement in one document and non-repudiation and security audit requirements in another document are all about collecting and keeping tamper proof records which can be used later to prevent users from denying their usages. It is difficult to understand the requirements specified by other parties using non-unified vocabularies. Misunderstanding of the requirements is a major cause for errors.

Secondly, composition can create unforeseen scenarios which are often missed. For example, the London airport terrorist plot last year could make deadly explosives through mixing several inert liquids, which appeared to be safe at security check all the time. Similarly, some operations of software systems may be safe if only performed within the component unit. But, they could cause devastating effects if combined with operations from other units.

Thirdly, since each of the stakeholders involved in collaborations has no prior knowledge on what requirements have/have not been specified by others when we

address all stakeholders' needs, some requirements may be requested more than once, or may not be satisfied at the same time.

In order to address these challenges in collaborative environments, the following techniques may be promising. A logic-based specification approach together with a domain specific ontology may provide unambiguous and interoperable specifications. Then, various logic reasoning analysis techniques may be applied. Situation awareness may be very effective in dealing with runtime composition in dynamic environments. Combining these techniques with powerful negotiation algorithms, compromises among stakeholders could be reached to reconcile possible conflicts.

Requirements Engineering Practice in the Development of a Bidding Decision Support System

Weicun Zhang [1], Lin Zhang [2]

1. Department of Automation
University of Science and Technology Beijing

Beijing 100083, P.R.China

E-mail:weicunzhang@263.net
2. School of Automation
Beijing University of Aeronautics and Astronautics
Beijing 100084, P.R.China

Abstract: This paper presents a requirement engineering practice in the development of a bidding decision support system which is useful for an independent power provider in electricity market. The system comprises eight subsystems: cost management, production management, forecast and decision, local data acquisition, market data acquisition, settlement management, bidding management and configuration management. The practical experiences and lessons are summarized from the feedback control perspectives.

Keywords: requirement engineering, electricity market, decision support system

1. INTRODUCTION

Many countries like Great Britain, United States, Australia, Norway etc., have introduced competition in power industry, and established electricity markets. There are lots of studies on such an issue [1-3]. With the introduction of electricity market, power industry is adopting e-business and its production mode will be gradually changed from planning mechanism to market mechanism. The power industry of China is also undergoing such reform. In some provinces, experimental markets have been put into trial use. To adapt to the market situation, a power plant or power company needs to strengthen its production management and cost management to get more benefits, meanwhile another way to increase income is to optimise its bidding strategy. Accordingly, a bidding decision support system is needed for these power providers. Generally speaking, a bidding decision support system should provide the decision maker the following information: production information, cost information, markets information (market clearing price information, power supply information and power demand information). Decision suggestions for all types of competition markets are essential contents of the system. Currently five markets are considered: real-time market, day-ahead market, forward contract market, automatic generation control (AGC) market and reserve market.

In the following sections, we will first give a general description of the bidding decision support system. Then we introduce the requirement engineering practice in the development of the software system with an experience summary from the perspectives of feedback control. Finally we depict the results of the application of the software system.

2. GENERAL DESCRIPTION OF THE SOFTWARE SYSTEM

2.1 Function Description

The goal of the bidding decision support system is to give the decision maker of an independent power provider an assistant or an advisor in the electricity market. Generally speaking, the decision maker needs to have a good command of the situation of the market, the production capability, the cost of generation. To be specific, the system should present all kinds of candidate bidding schedules and corresponding methodologies or indexes to the decision maker. Each schedule should also include the potential risk and potential benefit. Thus, the bidding decision support system should comprise eight subsystems: local data acquisition system (LDAS), market data acquisition system (MDAS), cost management system, production management system, forecast and decision system, settlement system, bidding management system and configuration management system. Figure 1 shows the functionalities of the system.

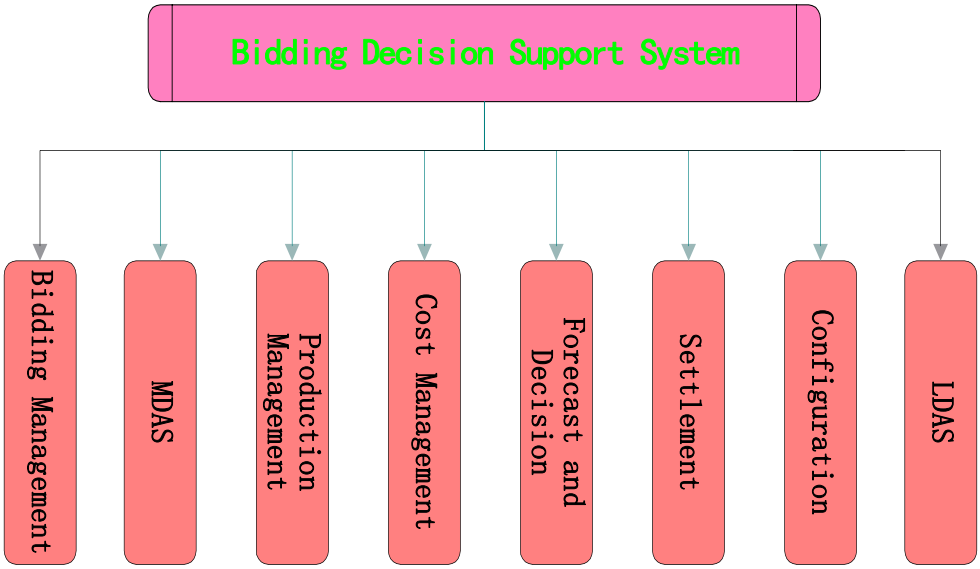


Figure 1 The function diagram of the bidding decision support system.

2.2 Data Flow and Logic Relations

From the point of view of data flow, the logic relation diagram of the software system is shown in Figure 2. First, the system gets basic local data of production and cost from distributed control system (DCS) and management information system (MIS) and other related information systems, also gets market data from open access same-time information system (OASIS). Second, the production management subsystem and cost management subsystem process those basic data and output useful parameters (like cost function and generation ability) to be used by market forecast and decision system. Finally, forecast and decision subsystem give the expected market price and bidding scheme suggestions. Settlement subsystem is interior for an independent power provider to check on the bill published by the market operator. Bidding management subsystem is designed for supervisors to adjust and finally approve the biddings. Each bidding scheme has its risk index and potential profit for reference. The functionalities of configuration

management include: user management, safety management and market rules management. For that the power market may change some operation rules (like trading time periods 24 ,48 or 96), the configuration management is needed to increase the adaptability of the bidding decision system.

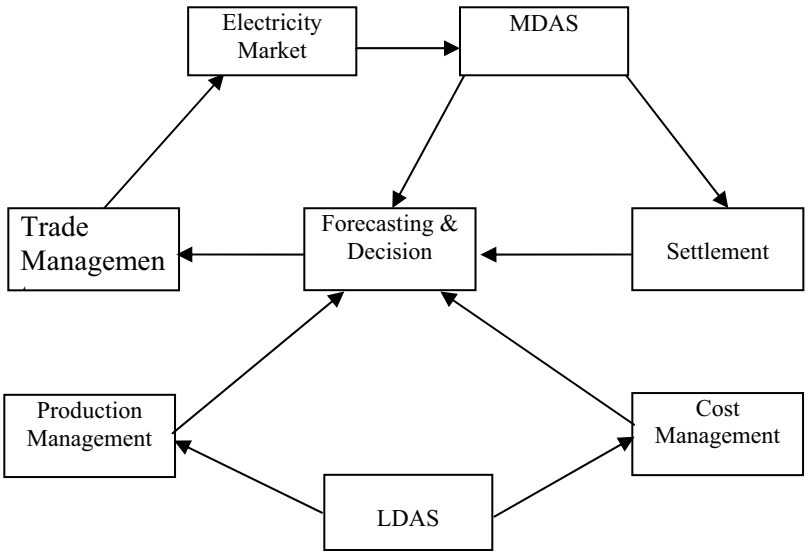


Figure 2 The logic relation diagram of the system

2.3 Core Functions

The core functions of the system are market price forecasting and bidding decision making. We developed 4 forecasting algorithms including: Forecast algorithm based on AR (Autoregressive) model; Forecast algorithm based on ARMA (Autoregressive Moving Average) model; Forecast algorithm based on ANN (Artificial Neural Network) model and Forecast algorithm based on dynamic fuzzy system model[7]. For decision making, we developed 3 schemes. They are: decision algorithm based on game theory [8]; decision algorithm based on demanded profit; decision algorithm based on break-even point. From input-output point of view, the bidding decision process can be described in Figure 3.

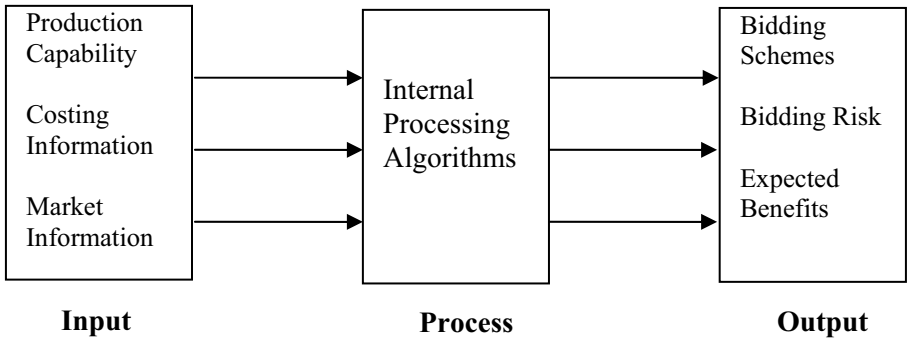


Figure 3 IPO diagram of the bidding decision function

2.4 Development Environment

The software structures are based on J2EE standards and Browser/Server mode. The development tools include JBuilder 9.0, Oracle 9i, and Rational ClearCase. The database server and application server are running on operating system of True 64 UNIX. And the operating system on terminals is MS Windows 2000.

3. REQUIREMENTS ENGINEERING OF THE SOFTWARE DEVELOPMENT

3.1 General Description

Requirement engineering (RE) is a multi-disciplinary activity, deploying a variety of techniques and tools at different stages of development and for different kinds of application domains [4]. A variety of approaches have been suggested to manage and integrate different RE activities and products [5, 6].

The development process, especially the RE process of this bidding decision support system is a little different from normal softwares. That is because we didn't receive any commitment before we started to develop the software. We just did it by our judgement of potential market needs, And in the mid of development we fortunately got our first commitment contract from a power plant. Before the contract was signed, there are four principal stages of RE, they are internal design of requirement specification, prototyping software, technical clarification & discussion meetings, requirement specification book. And after the contract was signed, there is one principal stage of RE, that is requirement change management. The RE process can be described by Figure 4.

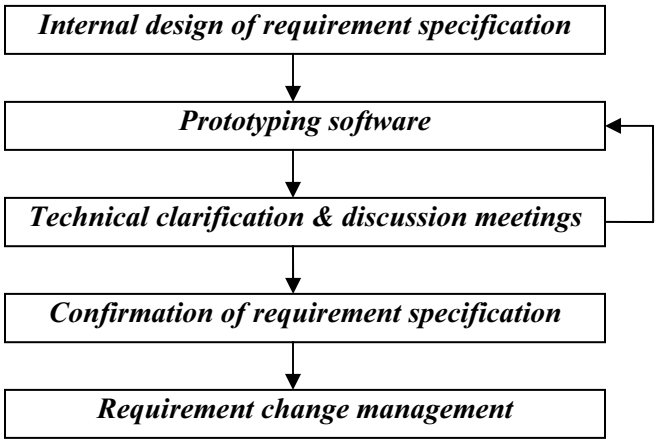


Figure 4 The process of requirement engineering

From the point of view of feedback control, the RE process can be also described in Figure 5. Requirement engineering is the "controller" in the software production process, which includes human being's intelligence, knowledge and relative tools to understand and express the stakeholder's explicit or inexplicit requirements. Some principles or methodologies, such as filtering, excitation and identification are useful for reference in requirement engineering.

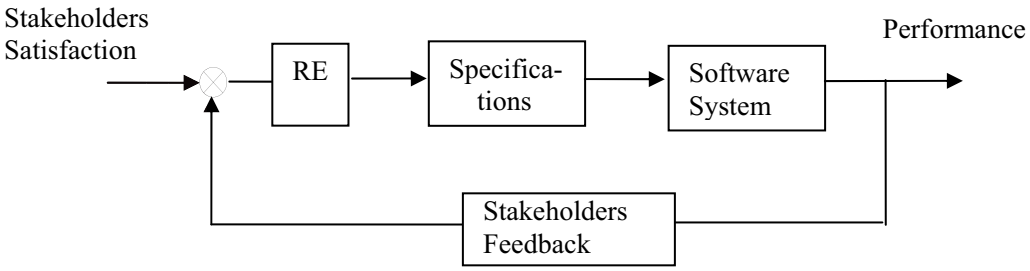


Figure 5 The feedback structure for requirement engineering

3.2 Internal design of requirement specification

According to our understanding of electricity market and power plant, we conceive the functionalities and data flows. Although a lot of changes had been made on the requirement specifications through the whole process of the bidding decision support software development, the structure of its eight subsystems and their logic relations remained the same as our initial design. The eight subsystems are: local data acquisition, market data acquisition, cost management, production management, forecast and decision, internal settlement, bidding management and configuration management.

3.3 Prototyping software system

After the accomplishment of internal design of requirement specifications, we started to develop the prototype software of the conceived system. The prototype software system was focused on the human machine interface (HMI) without database connection. That was the basic version of the software system. The development environments were the same as the real system. The prototype software system was mainly used for elicitation where there is a great deal of uncertainty and ambiguity about the requirements of stakeholders, or where early feedback from stakeholders is needed. Prototype software system can also be readily combined with other techniques, for instance by using a prototype to provoke discussion in a group elicitation technique.

3.4 Technical clarification & discussion meetings

After potential customer was found, we held technical liaison meetings several times, to improve the requirement specification and prototype system. Based on the prototype system, the customer gave many suggestions about the human-machine interfaces and other functionalities. The focused issue is local data acquisition. That was because that many local data had already been stored in different databases of different information systems of the independent power provider (or power plant), like management information system (also known as MIS), financial management information system (also known as FMIS), distributed control system (also known as DCS), production management system, telemeter reading system (also known as TMR system) and supervisory control and data acquisition system (also known as SCADA system). The difficulty also rested with that different software systems were developed by different vendor and different database were adopted. There were of course many overlapping of data definitions. Finally, we designed two kinds of interfaces for data acquisition. One way is to collect data on human-machine interface. Another way is to use the gateway program to convert data from existing databases to the database of the bidding decision support system.

3.5 Confirmation of requirement specification

After the stakeholder was basically satisfied, we had the requirement specification checked and accepted by the customer formally. This version of the requirement specification was a benchmark for later changes management. Actually we especially designed a confirmation book according to the requirement specification. In which, each specific functionality on each human-machine interface needs checks and signatures of both software developer and stakeholder. Figure 6 gives an example of such kind of confirmation forms.

Function	Operation log management	
Description	Management of system operation log	
Operations	Generate、modify、delete、print	
Log content (Please check term by term)		
Operator name		
Access date		
Access time		
Computer name		
System module name		
Sub-module name		
Customer Name Signature_____		Vendor Name Signature_____

Figure 6 An example of confirmation form of requirement specification

3.6 Requirement change management

We formulated specific criterion on requirement change management. Also, a requirement change form was prepared. If there is any change of requirements after contract was signed, a copy of change form should be filled by both sides with signatures. With the change form signature we wanted to control the quantity of changes, because sometimes the customer put forward change demands ad libitum. After the customer's requirement changes had been confirmed, we started another process, which is software change control. We formulated a procedure for software change control. There were mainly five steps for software change control: proposal of change, appraisal of change, auditing of change, approval of change and implementation of change.

3.7 Lessons and experiences

Sometimes the stakeholders requirements are explicit and specific, sometimes are inexplicit. The requirement engineering process could be described as a feedback control system. Our goal is to make the process (control system) stable, exact and quick. The key point lies in the "controller", i.e. the requirement engineering which comprises human intelligence and knowledge to understand the explicit and inexplicit requirements of the stakeholder. Sometimes we need to filter their frequent changes in requirement expression and linguistic ambiguity with the help of discussion meeting and prototype system, and sometimes we need to elicit or excite their genuine requirements. This is similar to the principle of system identification and control. The stakeholders usually don not care very much about the internal or background processing algorithm, like market price forecasting algorithms and decision making algorithms which plays the essential role in the bidding decision support system.

4. CONCLUDING REMARKS

The development of the bidding decision support system had been finished on time. And the software system is running well in some power plants. Although the economic benefit can not be evaluated up to now because the electricity market is still in its trial use stage in China, what we can see is that the software system can indeed provide reasonable bidding decisions under the trial use circumstances. And the costing management of the power plant is more specific and legible than before. The key to the success of the software development is that we noticed the importance of requirement engineering and adopted some corresponding measures. In summary, we present our understandings of requirement engineering from the viewpoint of feedback control, which may help improve the efficiency of requirement engineering.

REFERENCES

[1] Schwepp F C *et al*, Spot Price of Electricity. Kluwer Academic Publishers, 1988
[2] Hung-po Chao *et a*l, Designing Competitive Electricity Markets, Kluwer Academic Publishers, 1998
[3] Gerald B.Sheble, Computational Auction Mechanisms for Restructured Power Industry Operation, Kluwer Academic Publishers, 1999
[4] Bashar Nuseibeh, Steve Easterbrook, Requirements Engineering: A Roadmap, "The Future of Software Engineering" , Anthony Finkelstein (Ed.), ACM 2000.
[5] Jackson, M., Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices. Addison Wesley, 1995
[6] Aybüke Aurum, Claes. Wohlin (Eds.), Engineering and Managing Software Requirements, Springer, 2005
[7] Liu Hongjie, Wang Xiufeng, Zhang Weicun, and Xu. Guohua, Market clearing price forecasting based on dynamic fuzzy system. Proceedings of 2002 Power System Technology, Vol.2. pp.890-896
[8] Shang Jincheng, Huang Yonghao, Zhang Weicun, et al. A model and algorithm of game theory based bidding strategy for an independent power provider. Automation of Electric Power Systems, 2002, 26(9): pp.7-11.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Ft. Belvoir, Virginia	1
2.	Dudley Knox Library Naval Postgraduate School Monterey, California	1
3.	Dr. Joseph Olive DARPA/ IPTO Arlington, VA	3
4.	Dr. Helen Gill NSF/ Directorate for Computer and Information Science Arlington, VA	3
5.	Luqi Naval Postgraduate School Monterey, CA 4	3
6.	Craig Martell Naval Postgraduate School Monterey, CA	3
7.	George Dinolt Naval Postgraduate School Monterey, CA	1